CLICK HERE

Mathematics of the Discrete Fourier Transform (DFT)

Julius O. Smith III (jos@ccrma.stanford.edu)
Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305

March 15, 2002

Contents

1 Introduction to the DFT **1**
- 1.1 DFT Definition . . . 1
- 1.2 Mathematics of the DFT . . . 3
- 1.3 DFT Math Outline . . . 6

2 Complex Numbers **7**
- 2.1 Factoring a Polynomial . . . 7
- 2.2 The Quadratic Formula . . . 8
- 2.3 Complex Roots . . . 9
- 2.4 Fundamental Theorem of Algebra . . . 11
- 2.5 Complex Basics . . . 11
  - 2.5.1 The Complex Plane . . . 13
  - 2.5.2 More Notation and Terminology . . . 14
  - 2.5.3 Elementary Relationships . . . 15
  - 2.5.4 Euler's Formula . . . 15
  - 2.5.5 De Moivre's Theorem . . . 17
- 2.6 Numerical Tools in Matlab . . . 17
- 2.7 Numerical Tools in Mathematica . . . 23

3 Proof of Euler's Identity **27**
- 3.1 Euler's Theorem . . . 27
  - 3.1.1 Positive Integer Exponents . . . 27
  - 3.1.2 Properties of Exponents . . . 28
  - 3.1.3 The Exponent Zero . . . 28
  - 3.1.4 Negative Exponents . . . 28
  - 3.1.5 Rational Exponents . . . 29
  - 3.1.6 Real Exponents . . . 30
  - 3.1.7 A First Look at Taylor Series . . . 31
  - 3.1.8 Imaginary Exponents . . . 32

3.1.9 Derivatives of $f(x) = a^x$ 32
3.1.10 Back to e 33
3.1.11 Sidebar on Mathematica 34
3.1.12 Back to $e^{j\theta}$ 34
3.2 Informal Derivation of Taylor Series 36
3.3 Taylor Series with Remainder 37
3.4 Formal Statement of Taylor's Theorem 39
3.5 Weierstrass Approximation Theorem 40
3.6 Differentiability of Audio Signals 40

4 Logarithms, Decibels, and Number Systems 41
4.1 Logarithms 41
4.1.1 Changing the Base 43
4.1.2 Logarithms of Negative and Imaginary Numbers . 43
4.2 Decibels 44
4.2.1 Properties of DB Scales 45
4.2.2 Specific DB Scales 46
4.2.3 Dynamic Range 52
4.3 Linear Number Systems for Digital Audio 53
4.3.1 Pulse Code Modulation (PCM) 53
4.3.2 Binary Integer Fixed-Point Numbers 53
4.3.3 Fractional Binary Fixed-Point Numbers 58
4.3.4 How Many Bits are Enough for Digital Audio? . . 58
4.3.5 When Do We Have to Swap Bytes? 59
4.4 Logarithmic Number Systems for Audio 61
4.4.1 Floating-Point Numbers 61
4.4.2 Logarithmic Fixed-Point Numbers 63
4.4.3 Mu-Law Companding 64
4.5 Appendix A: Round-Off Error Variance 65
4.6 Appendix B: Electrical Engineering 101 66

5 Sinusoids and Exponentials 69
5.1 Sinusoids 69
5.1.1 Example Sinusoids 70
5.1.2 Why Sinusoids are Important 71
5.1.3 In-Phase and Quadrature Sinusoidal Components . 72
5.1.4 Sinusoids at the Same Frequency 73
5.1.5 Constructive and Destructive Interference 74
5.2 Exponentials 76

5.2.1 Why Exponentials are Important 77
5.2.2 Audio Decay Time (T60) 78
5.3 Complex Sinusoids 78
5.3.1 Circular Motion 79
5.3.2 Projection of Circular Motion 79
5.3.3 Positive and Negative Frequencies 80
5.3.4 The Analytic Signal and Hilbert Transform Filters 81
5.3.5 Generalized Complex Sinusoids 85
5.3.6 Sampled Sinusoids 86
5.3.7 Powers of z 86
5.3.8 Phasor & Carrier Components of Complex Sinusoids 87
5.3.9 Why Generalized Complex Sinusoids are Important 89
5.3.10 Comparing Analog and Digital Complex Planes . . 91
5.4 Mathematica for Selected Plots 94
5.5 Acknowledgement 95

6 Geometric Signal Theory 97
6.1 The DFT 97
6.2 Signals as Vectors 98
6.3 Vector Addition 99
6.4 Vector Subtraction 100
6.5 Signal Metrics 100
6.6 The Inner Product 105
6.6.1 Linearity of the Inner Product 106
6.6.2 Norm Induced by the Inner Product 107
6.6.3 Cauchy-Schwarz Inequality 107
6.6.4 Triangle Inequality 108
6.6.5 Triangle Difference Inequality 109
6.6.6 Vector Cosine 109
6.6.7 Orthogonality 109
6.6.8 The Pythagorean Theorem in N-Space 110
6.6.9 Projection 111
6.7 Signal Reconstruction from Projections 111
6.7.1 An Example of Changing Coordinates in 2D . . . 113
6.7.2 General Conditions 115
6.7.3 Gram-Schmidt Orthogonalization 119
6.8 Appendix: Matlab Examples 120

7 Derivation of the Discrete Fourier Transform (DFT) 127
7.1 The DFT Derived . . . 127
7.1.1 Geometric Series . . . 127
7.1.2 Orthogonality of Sinusoids . . . 128
7.1.3 Orthogonality of the DFT Sinusoids . . . 131
7.1.4 Norm of the DFT Sinusoids . . . 131
7.1.5 An Orthonormal Sinusoidal Set . . . 131
7.1.6 The Discrete Fourier Transform (DFT) . . . 132
7.1.7 Frequencies in the "Cracks" . . . 133
7.1.8 Normalized DFT . . . 136
7.2 The Length 2 DFT . . . 137
7.3 Matrix Formulation of the DFT . . . 138
7.4 Matlab Examples . . . 140
7.4.1 Figure 7.2 . . . 140
7.4.2 Figure 7.3 . . . 141
7.4.3 DFT Matrix in Matlab . . . 142

8 Fourier Theorems for the DFT 145
8.1 The DFT and its Inverse . . . 145
8.1.1 Notation and Terminology . . . 146
8.1.2 Modulo Indexing, Periodic Extension . . . 146
8.2 Signal Operators . . . 148
8.2.1 Flip Operator . . . 148
8.2.2 Shift Operator . . . 148
8.2.3 Convolution . . . 151
8.2.4 Correlation . . . 154
8.2.5 Stretch Operator . . . 155
8.2.6 Zero Padding . . . 155
8.2.7 Repeat Operator . . . 156
8.2.8 Downsampling Operator . . . 158
8.2.9 Alias Operator . . . 160
8.3 Even and Odd Functions . . . 163
8.4 The Fourier Theorems . . . 165
8.4.1 Linearity . . . 165
8.4.2 Conjugation and Reversal . . . 166
8.4.3 Symmetry . . . 167
8.4.4 Shift Theorem . . . 169
8.4.5 Convolution Theorem . . . 171
8.4.6 Dual of the Convolution Theorem . . . 173

8.4.7 Correlation Theorem 173
8.4.8 Power Theorem 174
8.4.9 Rayleigh Energy Theorem (Parseval's Theorem) . 174
8.4.10 Stretch Theorem (Repeat Theorem) 175
8.4.11 Downsampling Theorem (Aliasing Theorem) . . . 175
8.4.12 Zero Padding Theorem 177
8.4.13 Bandlimited Interpolation in Time 178
8.5 Conclusions 179
8.6 Acknowledgement 179
8.7 Appendix A: Linear Time-Invariant Filters and Convolution 180
8.7.1 LTI Filters and the Convolution Theorem 181
8.8 Appendix B: Statistical Signal Processing 182
8.8.1 Cross-Correlation 182
8.8.2 Applications of Cross-Correlation 183
8.8.3 Autocorrelation 186
8.8.4 Coherence 187
8.9 Appendix C: The Similarity Theorem 188

9 Example Applications of the DFT 191
9.1 Spectrum Analysis of a Sinusoid: Windowing, Zero-Padding, and the FFT 191
9.1.1 Example 1: FFT of a Simple Sinusoid 191
9.1.2 Example 2: FFT of a Not-So-Simple Sinusoid . . . 194
9.1.3 Example 3: FFT of a Zero-Padded Sinusoid 197
9.1.4 Example 4: Blackman Window 199
9.1.5 Example 5: Use of the Blackman Window 201
9.1.6 Example 6: Hanning-Windowed Complex Sinusoid 203

A Matrices 211
A.0.1 Matrix Multiplication 212
A.0.2 Solving Linear Equations Using Matrices 215

B Sampling Theory 217
B.1 Introduction 217
B.1.1 Reconstruction from Samples—Pictorial Version . 218
B.1.2 Reconstruction from Samples—The Math 219
B.2 Aliasing of Sampled Signals 220
B.3 Shannon's Sampling Theorem 223
B.4 Another Path to Sampling Theory 225

B.4.1 What frequencies are representable by a geometric sequence? 226
B.4.2 Recovering a Continuous Signal from its Samples . 228

Preface

This reader is an outgrowth of my course entitled "Introduction to Digital Signal Processing and the Discrete Fourier Transform (DFT)[1] which I have given at the Center for Computer Research in Music and Acoustics (CCRMA) every year for the past 16 years. The course was created primarily as a first course in digital signal processing for entering Music Ph.D. students. As a result, the only prerequisite is a good high-school math background. Calculus exposure is desirable, but not required.

Outline

Below is an overview of the chapters.

- **Introduction to the DFT**
  This chapter introduces the Discrete Fourier Transform (DFT) and points out the elements which will be discussed in this reader.
- **Introduction to Complex Numbers**
  This chapter provides an introduction to complex numbers, factoring polynomials, the quadratic formula, the complex plane, Euler's formula, and an overview of numerical facilities for complex numbers in Matlab and Mathematica.
- **Proof of Euler's Identity**
  This chapter outlines the proof of Euler's Identity, which is an important tool for working with complex numbers. It is one of the critical elements of the DFT definition that we need to understand.
- **Logarithms, Decibels, and Number Systems**
  This chapter discusses logarithms (real and complex), decibels, and

[1] http://www-ccrma.stanford.edu/CCRMA/Courses/320/

number systems such as binary integer fixed-point, fractional fixed-point, one's complement, two's complement, logarithmic fixed-point, μ-law, and floating-point number formats.

- **Sinusoids and Exponentials**
  This chapter provides an introduction to sinusoids, exponentials, complex sinusoids, t_{60}, in-phase and quadrature sinusoidal components, the analytic signal, positive and negative frequencies, constructive and destructive interference, invariance of sinusoidal frequency in linear time-invariant systems, circular motion as the vector sum of in-phase and quadrature sinusoidal motions, sampled sinusoids, generating sampled sinusoids from powers of z, and plot examples using Mathematica.

- **The Discrete Fourier Transform (DFT) Derived**
  This chapter derives the Discrete Fourier Transform (DFT) as a projection of a length N signal $x(\cdot)$ onto the set of N sampled complex sinusoids generated by the N roots of unity.

- **Fourier Theorems for the DFT**
  This chapter derives various *Fourier theorems* for the case of the DFT. Included are symmetry relations, the shift theorem, convolution theorem, correlation theorem, power theorem, and theorems pertaining to interpolation and downsampling. Applications related to certain theorems are outlined, including linear time-invariant filtering, sampling rate conversion, and statistical signal processing.

- **Example Applications of the DFT**
  This chapter goes through some practical examples of FFT analysis in Matlab. The various Fourier theorems provide a "thinking vocabulary" for understanding elements of spectral analysis.

- **A Basic Tutorial on Sampling Theory**
  This appendix provides a basic tutorial on sampling theory. Aliasing due to sampling of continuous-time signals is characterized mathematically. Shannon's sampling theorem is proved. A pictorial representation of continuous-time signal reconstruction from discrete-time samples is given.

Chapter 1

Introduction to the DFT

This chapter introduces the Discrete Fourier Transform (DFT) and points out the elements which will be discussed in this reader.

1.1 DFT Definition

The *Discrete Fourier Transform (DFT)* of a signal x may be defined by

$$X(\omega_k) \stackrel{\Delta}{=} \sum_{n=0}^{N-1} x(t_n) e^{-j\omega_k t_n}, \qquad k = 0, 1, 2, \ldots, N-1$$

and its *inverse* (the IDFT) is given by

$$x(t_n) = \frac{1}{N} \sum_{k=0}^{N-1} X(\omega_k) e^{j\omega_k t_n}, \qquad n = 0, 1, 2, \ldots, N-1$$

where

$$
\begin{aligned}
x(t_n) &\triangleq \text{input signal } \textit{amplitude} \text{ at time } t_n \text{ (sec)} \\
t_n &\triangleq nT = n\text{th sampling instant (sec)} \\
n &\triangleq \text{sample number (integer)} \\
T &\triangleq \text{sampling period (sec)} \\
X(\omega_k) &\triangleq \textit{Spectrum} \text{ of } x, \text{ at radian frequency } \omega_k \\
\omega_k &\triangleq k\Omega = k\text{th frequency sample (rad/sec)} \\
\Omega &\triangleq \frac{2\pi}{NT} = \text{radian-frequency sampling interval} \\
f_s &\triangleq 1/T = \textit{sampling rate} \text{ (samples/sec, or Hertz (Hz))} \\
N &= \text{number of samples in both time and frequency (integer)}
\end{aligned}
$$

1.2 Mathematics of the DFT

In the signal processing literature, it is common to write the DFT in the more pure form obtained by setting $T = 1$ in the previous definition:

$$\begin{aligned} X(k) &\stackrel{\Delta}{=} \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N}, \qquad k = 0, 1, 2, \ldots, N-1 \\ x(n) &= \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi nk/N}, \qquad n = 0, 1, 2, \ldots, N-1 \end{aligned}$$

where $x(n)$ denotes the input signal at time (sample) n, and $X(k)$ denotes the kth spectral sample.[1] This form is the simplest mathematically while the previous form is the easier to interpret physically.

There are two remaining symbols in the DFT that we have not yet defined:

$$\begin{aligned} j &\stackrel{\Delta}{=} \sqrt{-1} \\ e &\stackrel{\Delta}{=} \lim_{n\to\infty}\left(1 + \frac{1}{n}\right)^n = 2.71828182845905\ldots \end{aligned}$$

The first, $j = \sqrt{-1}$, is the basis for *complex numbers*. As a result, complex numbers will be the first topic we cover in this reader (but only to the extent needed to understand the DFT).

The second, $e = 2.718\ldots$, is a transcendental number defined by the above limit. In this reader, we will derive e and talk about why it comes up.

Note that not only do we have complex numbers to contend with, but we have them appearing in exponents, as in

$$s_k(n) \stackrel{\Delta}{=} e^{j2\pi nk/N}$$

We will systematically develop what we mean by imaginary exponents in order that such mathematical expressions are well defined.

With e, j, and imaginary exponents understood, we can go on to prove *Euler's Identity*:

$$e^{j\theta} = \cos(\theta) + j\sin(\theta)$$

[1] Note that the definition of $x()$ has changed unless the sampling rate f_s really is 1, and the definition of $X()$ has changed no matter what the sampling rate is, since when $T = 1$, $\omega_k = 2\pi k/N$, not k.

Euler's Identity is the key to understanding the meaning of expressions like

$$s_k(t_n) \stackrel{\Delta}{=} e^{j\omega_k t_n} = \cos(\omega_k t_n) + j\sin(\omega_k t_n)$$

We'll see that such an expression defines a *sampled complex sinusoid*, and we'll talk about sinusoids in some detail, from an audio perspective.

Finally, we need to understand what the summation over n is doing in the definition of the DFT. We'll learn that it should be seen as the computation of the *inner product* of the signals x and s_k, so that we may write the DFT using inner-product notation as

$$X(k) \stackrel{\Delta}{=} \langle x, s_k \rangle$$

where

$$s_k(n) \stackrel{\Delta}{=} e^{j2\pi nk/N}$$

is the sampled complex sinusoid at (normalized) radian frequency $\omega_k = 2\pi k/N$, and the inner product operation is defined by

$$\langle x, y \rangle \stackrel{\Delta}{=} \sum_{n=0}^{N-1} x(n)\overline{y(n)}$$

We will show that the inner product of x with the kth "basis sinusoid" s_k is a measure of "how much" of s_k is present in x and at "what phase" (since it is a complex number).

After the foregoing, the inverse DFT can be understood as the *weighted sum of projections* of x onto $\{s_k\}_{k=0}^{N-1}$, i.e.,

$$x(n) \stackrel{\Delta}{=} \sum_{k=0}^{N-1} \tilde{X}_k s_k(n)$$

where

$$\tilde{X}_k \stackrel{\Delta}{=} \frac{X(k)}{N}$$

is the (actual) *coefficient of projection* of x onto s_k. Referring to the whole signal as $x \stackrel{\Delta}{=} x(\cdot)$, the IDFT can be written as

$$x \stackrel{\Delta}{=} \sum_{k=0}^{N-1} \tilde{X}_k s_k$$

Note that both the *basis sinusoids* s_k and their coefficients of projection $\tilde{X}_k$ are *complex*.

Having completely understood the DFT and its inverse mathematically, we go on to proving various *Fourier Theorems*, such as the "shift theorem," the "convolution theorem," and "Parseval's theorem." The Fourier theorems provide a basic thinking vocabulary for working with signals in the time and frequency domains. They can be used to answer questions like

> What happens in the frequency domain if I do [x] in the time domain?

Finally, we will study a variety of practical spectrum analysis examples, using primarily Matlab to analyze and display signals and their spectra.

1.3 DFT Math Outline

In summary, understanding the DFT takes us through the following topics:

1. Complex numbers
2. Complex exponents
3. Why e?
4. Euler's formula
5. Projecting signals onto signals via the inner product
6. The DFT as the coefficient of projection of a signal x onto a sinusoid
7. The IDFT as a weighted sum of sinusoidal projections
8. Various Fourier theorems
9. Elementary time-frequency pairs
10. Practical spectrum analysis in matlab

We will additionally discuss practical aspects of working with sinusoids, such as *decibels* (dB) and display techniques.

Chapter 2

Complex Numbers

This chapter provides an introduction to complex numbers, factoring polynomials, the quadratic formula, the complex plane, Euler's formula, and an overview of numerical facilities for complex numbers in Matlab and Mathematica.

2.1 Factoring a Polynomial

Remember "factoring polynomials"? Consider the second-order polynomial

$$p(x) = x^2 - 5x + 6$$

It is second-order because the highest power of x is 2 (only non-negative integer powers of x are allowed in this context). The polynomial is also *monic* because its leading coefficient, the coefficient of x^2, is 1. Since it is second order, there are at most two real *roots* (or *zeros*) of the polynomial. Suppose they are denoted x_1 and x_2. Then we have $p(x_1) = 0$ and $p(x_2) = 0$, and we can write

$$p(x) = (x - x_1)(x - x_2)$$

This is the *factored form* of the monic polynomial $p(x)$. (For a non-monic polynomial, we may simply divide all coefficients by the first to make it monic, and this doesn't affect the zeros.) Multiplying out the symbolic factored form gives

$$p(x) = (x - x_1)(x - x_2) = x^2 - (x_1 + x_2)x + x_1x_2$$

Comparing with the original polynomial, we find we must have

$$\begin{aligned} x_1 + x_2 &= 5 \\ x_1 x_2 &= 6 \end{aligned}$$

This is a system of two equations in two unknowns. Unfortunately, it is a *nonlinear* system of two equations in two unknowns.[1] Nevertheless, because it is so small, the equations are easily solved. In beginning algebra, we did them by hand. However, nowadays we can use a computer program such as Mathematica:

```
In[]:=
        Solve[{x1+x2==5, x1 x2 == 6}, {x1,x2}]
Out[]:
        {{x1 -> 2, x2 -> 3}, {x1 -> 3, x2 -> 2}}
```

Note that the two lists of substitutions point out that it doesn't matter which root is 2 and which is 3. In summary, the factored form of this simple example is

$$p(x) = x^2 - 5x + 6 = (x - x_1)(x - x_2) = (x - 2)(x - 3)$$

Note that polynomial factorization rewrites a monic nth-order polynomial as the product of n *first-order* monic polynomials, each of which contributes one zero (root) to the product. This factoring business is often used when working with *digital filters*.

2.2 The Quadratic Formula

The general second-order polynomial is

$$p(x) \stackrel{\Delta}{=} ax^2 + bx + c$$

where the coefficients a, b, c are any real numbers, and we assume $a \neq 0$ since otherwise it would not be second order. Some experiments plotting

[1] "Linear" in this context means that the unknowns are multiplied only by constants—they may not be multiplied by each other or raised to any power other than 1 (e.g., not squared or cubed or raised to the 1/5 power). Linear systems of N equations in N unknowns are very easy to solve compared to *nonlinear* systems of N equations in N unknowns. For example, Matlab or Mathematica can easily handle them. You learn all about this in a course on *Linear Algebra* which is highly recommended for anyone interested in getting involved with signal processing. Linear algebra also teaches you all about *matrices* which we will introduce only briefly in this reader.

$p(x)$ for different values of the coefficients leads one to guess that the curve is always a scaled and translated *parabola*. The canonical parabola centered at $x = x_0$ is given by

$$y(x) = d(x - x_0)^2 + e$$

where d determines the width and e provides an arbitrary vertical offset. If we can find d, e, x_0 in terms of a, b, c for any quadratic polynomial, then we can easily factor the polynomial. This is called "completing the square." Multiplying out $y(x)$, we get

$$y(x) = d(x - x_0)^2 + e = dx^2 - 2dx_0x + dx_0^2 + e$$

Equating coefficients of like powers of x gives

$$\begin{aligned} d &= a \\ -2dx_0 &= b \quad \Rightarrow \quad x_0 = -b/(2a) \\ dx_0^2 + e &= c \quad \Rightarrow \quad e = c - b^2/(4a) \end{aligned}$$

Using these answers, any second-order polynomial $p(x) = ax^2 + bx + c$ can be rewritten as a scaled, translated parabola

$$p(x) = a\left(x + \frac{b}{2a}\right)^2 + \left(c - \frac{b^2}{4a}\right).$$

In this form, the roots are easily found by solving $p(x) = 0$ to get

$$\boxed{x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}}$$

This is the general *quadratic formula*. It was obtained by simple algebraic manipulation of the original polynomial. There is only one "catch." What happens when $b^2 - 4ac$ is negative? This introduces the square root of a negative number which we could insist "does not exist." Alternatively, we could invent complex numbers to accommodate it.

2.3 Complex Roots

As a simple example, let $a = 1$, $b = 0$, and $c = 4$, i.e.,

$$p(x) = x^2 + 4$$

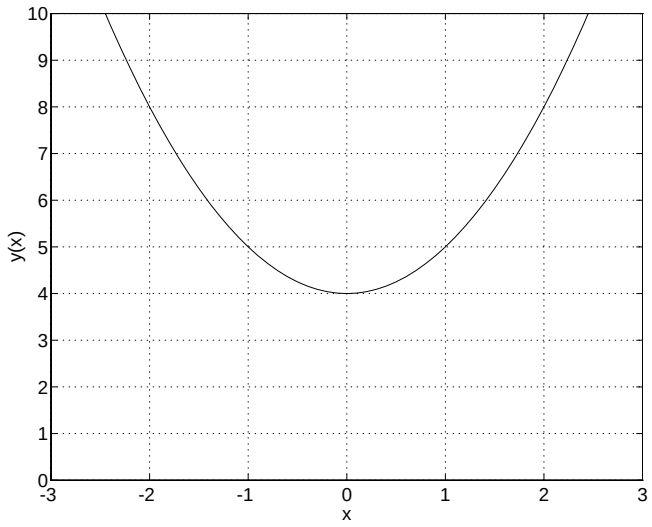

Figure 2.1: An example parabola defined by $p(x) = x^2 + 4$.

As shown in Fig. 2.1, this is a parabola centered at $x = 0$ (where $p(0) = 4$) and reaching upward to positive infinity, never going below 4. It has no zeros. On the other hand, the quadratic formula says that the "roots" are given formally by $x = \pm\sqrt{-4} = \pm 2\sqrt{-1}$. The square root of any negative number $c < 0$ can be expressed as $\sqrt{|c|}\sqrt{-1}$, so the only new algebraic object is $\sqrt{-1}$. Let's give it a name:

$$\boxed{j \triangleq \sqrt{-1}}$$

Then, formally, the roots of of x^2+4 are $\pm 2j$, and we can formally express the polynomial in terms of its roots as

$$p(x) = (x + 2j)(x - 2j)$$

We can think of these as "imaginary roots" in the sense that square roots of negative numbers don't really exist, or we can extend the concept of "roots" to allow for *complex numbers*, that is, numbers of the form

$$z = x + jy$$

where x and y are real numbers, and $j^2 \triangleq -1$.

It can be checked that all algebraic operations for real numbers[2] apply equally well to complex numers. Both real numbers and complex numbers are examples of a mathematical *field*. Fields are *closed* with respect to multiplication and addition, and all the rules of algebra we use in manipulating polynomials with real coefficients (and roots) carry over unchanged to polynomials with complex coefficients and roots. In fact, the rules of algebra become simpler for complex numbers because, as discussed in the next section, we can *always* factor polynomials completely over the field of complex numbers while we cannot do this over the reals (as we saw in the example $p(x) = x^2 + 4$).

2.4 Fundamental Theorem of Algebra

Every nth-order polynomial possesses exactly n complex roots.

This is a very powerful algebraic tool. It says that given any polynomial

$$\begin{aligned} p(x) &= a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_2 x^2 + a_1 x + a_0 \\ &\triangleq \sum_{i=0}^{n} a_i x^i \end{aligned}$$

we can *always* rewrite it as

$$\begin{aligned} p(x) &= a_n(x - z_n)(x - z_{n-1})(x - z_{n-2})\cdots(x - z_2)(x - z_1) \\ &\triangleq a_n \prod_{i=1}^{n}(x - z_i) \end{aligned}$$

where the points z_i are the polynomial roots, and they may be real or complex.

2.5 Complex Basics

This section introduces various notation and terms associated with complex numbers. As discussed above, complex numbers are devised by introducing the square-root of -1 as a primitive new algebraic object among

[2] multiplication, addition, division, distributivity of multiplication over addition, commutativity of multiplication and addition.

real numbers and manipulating it symbolically as if it were a real number itself:

$$\boxed{j \triangleq \sqrt{-1}}$$

Mathemeticians and physicists often use i instead of j as $\sqrt{-1}$. The use of j is common in engineering where i is more often used for electrical current.

As mentioned above, for any negative number $c < 0$, we have $\sqrt{c} = \sqrt{(-1)(-c)} = j\sqrt{|c|}$, where $|c|$ denotes the absolute value of c. Thus, every square root of a negative number can be expressed as j times the square root of a positive number.

By definition, we have

$$\begin{aligned} j^0 &= 1 \\ j^1 &= j \\ j^2 &= -1 \\ j^3 &= -j \\ j^4 &= 1 \\ &\cdots \end{aligned}$$

and so on. Thus, the sequence $x(n) \triangleq j^n$, $n = 0, 1, 2, \ldots$ is a periodic sequence with period 4, since $j^{n+4} = j^n j^4 = j^n$. (We'll learn later that the sequence j^n is a sampled complex sinusoid having frequency equal to one fourth the sampling rate.)

Every *complex number* z can be written as

$$\boxed{z = x + jy}$$

where x and y are real numbers. We call x the *real part* and y the *imaginary part*. We may also use the notation

$$\begin{aligned} \mathrm{re}\{z\} &= x \qquad (\text{"the real part of } z = x + jy \text{ is } x\text{"}) \\ \mathrm{im}\{z\} &= y \qquad (\text{"the imaginary part of } z = x + jy \text{ is } y\text{"}) \end{aligned}$$

Note that the real numbers are the subset of the complex numbers having a zero imaginary part ($y = 0$).

The rule for *complex multiplication* follows directly from the definition of the imaginary unit j:

$$\begin{aligned} z_1 z_2 &\triangleq (x_1 + jy_1)(x_2 + jy_2) \\ &= x_1 x_2 + j x_1 y_2 + j y_1 x_2 + j^2 y_1 y_2 \\ &= (x_1 x_2 - y_1 y_2) + j(x_1 y_2 + y_1 x_2) \end{aligned}$$

In some mathematics texts, complex numbers z are defined as ordered pairs of real numbers (x, y), and algebraic operations such as multiplication are defined more formally as operations on ordered pairs, e.g., $(x_1, y_1) \cdot (x_2, y_2) \stackrel{\Delta}{=} (x_1x_2 - y_1y_2, x_1y_2 + y_1x_2)$. However, such formality tends to obscure the underlying simplicity of complex numbers as a straightforward extension of real numbers to include $j \stackrel{\Delta}{=} \sqrt{-1}$.

It is important to realize that complex numbers can be treated algebraically just like real numbers. That is, they can be added, subtracted, multiplied, divided, etc., using exactly the same rules of algebra (since both real and complex numbers are mathematical *fields*). It is often preferable to think of complex numbers as being the true and proper setting for algebraic operations, with real numbers being the limited subset for which $y = 0$.

To explore further the magical world of complex variables, see any textbook such as [1, 2].

2.5.1 The Complex Plane

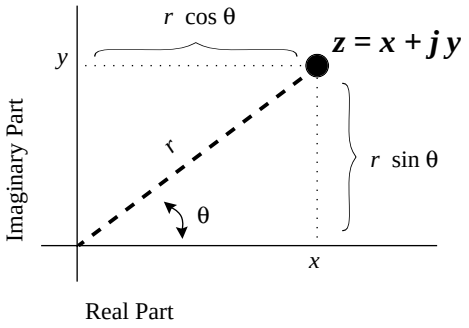

Figure 2.2: Plotting a complex number as a point in the complex plane.

We can plot any complex number $z = x + jy$ in a plane as an ordered pair (x, y), as shown in Fig. 2.2. A *complex plane* is any 2D graph in which the horizontal axis is the *real part* and the vertical axis is the *imaginary part* of a complex number or function. As an example, the number j has coordinates $(0, 1)$ in the complex plane while the number 1 has coordinates $(1, 0)$.

Plotting $z = x + jy$ as the point (x, y) in the complex plane can be

viewed as a plot in *Cartesian* or *rectilinear* coordinates. We can also express complex numbers in terms of *polar coordinates* as an ordered pair (r, θ), where r is the distance from the origin $(0, 0)$ to the number being plotted, and θ is the angle of the number relative to the positive real coordinate axis (the line defined by $y = 0$ and $x > 0$). (See Fig. 2.2.)

Using elementary *geometry*, it is quick to show that conversion from rectangular to polar coordinates is accomplished by the formulas

$$\boxed{\begin{aligned} r &= \sqrt{x^2 + y^2} \\ \theta &= \tan^{-1}(y/x). \end{aligned}}$$

The first equation follows immediately from the *Pythagorean theorem* , while the second follows immediately from the definition of the *tangent* function. Similarly, conversion from polar to rectangular coordinates is simply

$$\boxed{\begin{aligned} x &= r\ \cos(\theta) \\ y &= r\ \sin(\theta). \end{aligned}}$$

These follow immediately from the definitions of cosine and sine, respectively,

2.5.2 More Notation and Terminology

It's already been mentioned that the rectilinear coordinates of a complex number $z = x + jy$ in the complex plane are called the *real part* and *imaginary part*, respectively.

We also have special notation and various names for the radius and angle of a complex number z expressed in polar coordinates (r, θ):

$$\begin{aligned} r &\stackrel{\Delta}{=} |z| = \sqrt{x^2 + y^2} \\ &= \textit{modulus}, \textit{magnitude}, \textit{absolute value}, \textit{norm}, \text{ or } \textit{radius} \text{ of } z \\ \theta &\stackrel{\Delta}{=} \angle z = \tan^{-1}(y/x) \\ &= \textit{angle}, \textit{argument}, \text{ or } \textit{phase} \text{ of } z \end{aligned}$$

The *complex conjugate* of z is denoted $\overline{z}$ and is defined by

$$\overline{z} \stackrel{\Delta}{=} x - jy$$

where, of course, $z \stackrel{\Delta}{=} x + jy$. Sometimes you'll see the notation z^* in place of $\overline{z}$, but we won't use that here.

In general, you can always obtain the complex conjugate of any expression by simply replacing j with $-j$. In the complex plane, this is a *vertical flip* about the real axis; in other words, complex conjugation replaces each point in the complex plane by its *mirror image* on the other side of the x axis.

2.5.3 Elementary Relationships

From the above definitions, one can quickly verify

$$\begin{aligned} z + \overline{z} &= 2\,\mathrm{re}\{z\} \\ z - \overline{z} &= 2j\,\mathrm{im}\{z\} \\ z\overline{z} &= |z|^2 \end{aligned}$$

Let's verify the third relationship which states that a complex number multiplied by its conjugate is equal to its magnitude squared:

$$z\overline{z} \stackrel{\Delta}{=} (x + jy)(x - jy) = x^2 - (jy)^2 = x^2 + y^2 \stackrel{\Delta}{=} |z|^2,$$

2.5.4 Euler's Formula

Since $z = x + jy$ is the algebraic expression of z in terms of its rectangular coordinates, the corresponding expression in terms of its polar coordinates is

$$z = r\cos(\theta) + j\,r\sin(\theta).$$

There is another, more powerful representation of z in terms of its polar coordinates. In order to define it, we must introduce *Euler's Formula*:

$$\boxed{e^{j\theta} = \cos(\theta) + j\sin(\theta)} \tag{2.1}$$

A proof of Euler's identity is given in the next chapter. Just note for the moment that for $\theta = 0$, we have $e^{j0} = \cos(0) + j\sin(0) = 1 + j0 = 1$, as expected. Before, the only algebraic representation of a complex number we had was $z = x + jy$, which fundamentally uses Cartesian (rectilinear) coordinates in the complex plane. Euler's identity gives us an alternative algebraic representation in terms of polar coordinates in the complex plane:

$$\boxed{z = re^{j\theta}}$$

This representation often simplifies manipulations of complex numbers, especially when they are multiplied together. Simple rules of exponents can often be used in place of more difficult trigonometric identities. In the simple case of two complex numbers being multiplied,

$$z_1 z_2 = \left(r_1 e^{j\theta_1}\right)\left(r_2 e^{j\theta_2}\right) = (r_1 r_2)\left(e^{j\theta_1} e^{j\theta_2}\right) = r_1 r_2 e^{j(\theta_1+\theta_2)}$$

A corollary of Euler's identity is obtained by setting $\theta = \pi$ to get

$$e^{j\pi} + 1 = 0$$

This has been called the "most beautiful formula in mathematics" due to the extremely simple form in which the fundamental constants $e, j, \pi, 1$, and 0, together with the elementary operations of addition, multiplication, exponentiation, and equality, all appear exactly once.

For another example of manipulating the polar form of a complex number, let's again verify $z\overline{z} = |z|^2$, as we did above, but this time using polar form:

$$z\overline{z} = re^{j\theta} re^{-j\theta} = r^2 e^0 = r^2 = |z|^2.$$

We can now easily add a fourth line to that set of examples:

$$z/\overline{z} = \frac{re^{j\theta}}{re^{-j\theta}} = e^{j2\theta} = e^{j2\angle z}$$

Thus, $|z/\overline{z}| = 1$ for every $z \neq 0$.

Euler's identity can be used to derive formulas for sine and cosine in terms of $e^{j\theta}$:

$$\begin{aligned} e^{j\theta} + \overline{e^{j\theta}} &= e^{j\theta} + e^{-j\theta} \\ &= [\cos(\theta) + j\sin(\theta)] + [\cos(\theta) - j\sin(\theta)] \\ &= 2\cos(\theta), \end{aligned}$$

Similarly, $e^{j\theta} - \overline{e^{j\theta}} = 2j\ \sin(\theta)$, and we have

$$\boxed{\begin{aligned} \cos(\theta) &= \frac{e^{j\theta} + e^{-j\theta}}{2} \\ \sin(\theta) &= \frac{e^{j\theta} - e^{-j\theta}}{2j} \end{aligned}}$$

2.5.5 De Moivre's Theorem

As a more complicated example of the value of the polar form, we'll prove *De Moivre's theorem*:

$$[\cos(\theta) + j\sin(\theta)]^n = \cos(n\theta) + j\sin(n\theta)$$

Working this out using sum-of-angle identities from trigonometry is laborious. However, using Euler's identity, De Moivre's theorem simply "falls out":

$$[\cos(\theta) + j\sin(\theta)]^n = \left[e^{j\theta}\right]^n = e^{j\theta n} = \cos(n\theta) + j\sin(n\theta)$$

Moreover, by the power of the method used to show the result, n can be any real number, not just an integer.

2.6 Numerical Tools in Matlab

In Matlab, root-finding is always numerical:[3]

```
>> % polynomial = array of coefficients in Matlab
>> p = [1 0 0 0 5 7]; %  p(x) = x^5 + 5*x + 7
>> format long;       %  print double-precision
>> roots(p)           %  print out the roots of p(x)

ans =
  1.30051917307206 + 1.10944723819596i
  1.30051917307206 - 1.10944723819596i
 -0.75504792501755 + 1.27501061923774i
 -0.75504792501755 - 1.27501061923774i
 -1.09094249610903
```

Matlab has the following primitives for complex numbers:

```
>> help j

 J      Imaginary unit.
        The variables i and j both initially have the value sqrt(-1)
        for use in forming complex quantities.  For example, the
```

[3] unless you have the optional Maple package for symbolic mathematical manipulation

```
        expressions 3+2i, 3+2*i, 3+2i, 3+2*j and 3+2*sqrt(-1).
        all have the same value.  However, both i and j may be
        assigned other values, often in FOR loops and as subscripts.

        See also I.
        Built-in function.
        Copyright (c) 1984-92 by The MathWorks, Inc.

>> sqrt(-1)

ans =
        0 + 1.0000i

>> help real

 REAL   Complex real part.
        REAL(X) is the real part of X.

        See also IMAG, CONJ, ANGLE, ABS.

>> help imag

 IMAG   Complex imaginary part.
        IMAG(X) is the imaginary part of X.
        See I or J to enter complex numbers.

        See also REAL, CONJ, ANGLE, ABS.

>> help conj

 CONJ   Complex conjugate.
        CONJ(X) is the complex conjugate of X.

>> help abs

 ABS    Absolute value and string to numeric conversion.
        ABS(X) is the absolute value of the elements of X. When
        X is complex, ABS(X) is the complex modulus (magnitude) of
        the elements of X.
```

```
        See also ANGLE, UNWRAP.

        ABS(S), where S is a MATLAB string variable, returns the
        numeric values of the ASCII characters in the string.
        It does not change the internal representation, only the
        way it prints.
        See also SETSTR.

>> help angle

 ANGLE  Phase angle.
        ANGLE(H) returns the phase angles, in radians, of a matrix with
        complex elements.

        See also ABS, UNWRAP.
```

Note how helpful the "See also" information is in Matlab.

Let's run through a few elementary manipulations of complex numbers in Matlab:

```
>> x = 1; % Every symbol must have a value in Matlab
>> y = 2;
>> z = x + j * y

z =
   1.0000 + 2.0000i

>> 1/z

ans =
   0.2000 - 0.4000i

>> z^2

ans =
  -3.0000 + 4.0000i

>> conj(z)
```

```
ans =
   1.0000 - 2.0000i

>> z*conj(z)

ans =
     5

>> abs(z)^2

ans =
    5.0000

>> norm(z)^2

ans =
    5.0000

>> angle(z)

ans =
    1.1071
```

Now let's do polar form:

```
>> r = abs(z)

r =
    2.2361

>> theta = angle(z)

theta =
    1.1071
```

Curiously, e is not defined by default in Matlab (though it is in Octave). It can easily be computed in Matlab as `e=exp(1)`. Below are some examples involving imaginary exponentials:

```
>> r * exp(j * theta)

ans =
   1.0000 + 2.0000i

>> z

z =
   1.0000 + 2.0000i

>> z/abs(z)

ans =
   0.4472 + 0.8944i

>> exp(j*theta)

ans =
   0.4472 + 0.8944i

>> z/conj(z)

ans =
  -0.6000 + 0.8000i

>> exp(2*j*theta)

ans =
  -0.6000 + 0.8000i

>> imag(log(z/abs(z)))

ans =
    1.1071

>> theta

theta =
    1.1071
```

```
>>
```

Some manipulations involving two complex numbers:

```
>> x1 = 1;
>> x2 = 2;
>> y1 = 3;
>> y2 = 4;
>> z1 = x1 + j * y1;
>> z2 = x2 + j * y2;
>> z1

z1 =
   1.0000 + 3.0000i

>> z2

z2 =
   2.0000 + 4.0000i

>> z1*z2

ans =
 -10.0000 +10.0000i

>> z1/z2

ans =
   0.7000 + 0.1000i
```

Another thing to note about Matlab is that the transpose operator ' (for vectors and matrices) conjugates as well as transposes. Use .' to transpose without conjugation:

```
>>x = [1:4]*j

x =
        0 + 1.0000i        0 + 2.0000i        0 + 3.0000i        0 + 4.0000i

>> x'
```

```
ans =
        0 - 1.0000i
        0 - 2.0000i
        0 - 3.0000i
        0 - 4.0000i

>> x.'

ans =
        0 + 1.0000i
        0 + 2.0000i
        0 + 3.0000i
        0 + 4.0000i

>>
```

2.7 Numerical Tools in Mathematica

In Mathematica, we can find the roots of simple polynomials in closed form, while larger polynomials can be factored numerically in either Matlab or Mathematica. Look to Mathematica to provide the most sophisticated symbolic mathematical manipulation, and look for Matlab to provide the best numerical algorithms, as a general rule.

One way to implicitly find the roots of a polynomial is to *factor* it in Mathematica:

```
In[1]:
        p[x_] := x^2 + 5 x + 6
In[2]:
        Factor[p[x]]
Out[2]:
        (2 + x)*(3 + x)
```

Factor[] works only with exact Integers or Rational coefficients, not with Real numbers.

Alternatively, we can explicitly solve for the roots of low-order polynomials in Mathematica:

```
In[1]:
```

```
        p[x_] := a x^2 + b x + c
In[2]:
        Solve[p[x]==0,x]
Out[2]:
        {{x -> (-(b/a) + (b^2/a^2 - (4*c)/a)^(1/2))/2},
         {x -> (-(b/a) - (b^2/a^2 - (4*c)/a)^(1/2))/2}}
```

Closed-form solutions work for polynomials of order one through four. Higher orders, in general, must be dealt with numerically, as shown below:

```
In[1]:
        p[x_] := x^5 + 5 x + 7
In[2]:
        Solve[p[x]==0,x]
Out[2]:
        {ToRules[Roots[5*x + x^5 == -7, x]]}
In[3]:
        N[Solve[p[x]==0,x]]
Out[3]:
        {{x -> -1.090942496109028},
          {x -> -0.7550479250175501 - 1.275010619237742*I},
          {x -> -0.7550479250175501 + 1.275010619237742*I},
          {x -> 1.300519173072064 - 1.109447238195959*I},
          {x -> 1.300519173072064 + 1.109447238195959*I}}
```

Mathematica has the following primitives for dealing with complex numbers (The "?" operator returns a short description of the symbol to its right):

```
In[1]:
        ?I
Out[1]:
        I represents the imaginary unit Sqrt[-1].
In[2]:
        ?Re
Out[2]:
        Re[z] gives the real part of the complex number z.
In[3]:
        ?Im
Out[3]:
        Im[z] gives the imaginary part of the complex number z.
```

```
In[4]:
        ?Conj*
Out[4]:
        Conjugate[z] gives the complex conjugate of the complex number z.
In[5]:
        ?Abs
Out[5]:
        Abs[z] gives the absolute value of the real or complex number z.
In[6]:
        ?Arg
Out[6]:
        Arg[z] gives the argument of z.
```

Chapter 3

Proof of Euler's Identity

This chapter outlines the proof of Euler's Identity, which is an important tool for working with complex numbers. It is one of the critical elements of the DFT definition that we need to understand.

3.1 Euler's Theorem

Euler's Theorem (or "identity" or "formula") is

$$e^{j\theta} = \cos(\theta) + j\sin(\theta) \qquad \text{(Euler's Identity)}$$

To "prove" this, we must first define what we mean by "$e^{j\theta}$." (The right-hand side is assumed to be understood.) Since e is just a particular number, we only really have to explain what we mean by imaginary exponents. (We'll also see where e comes from in the process.) Imaginary exponents will be obtained as a generalization of real exponents. Therefore, our first task is to define exactly what we mean by a^x, where x is any real number, and $a > 0$ is any positive real number.

3.1.1 Positive Integer Exponents

The "original" definition of exponents which "actually makes sense" applies only to positive integer exponents:

$$\boxed{a^n \triangleq \underbrace{a \cdot a \cdot a \cdot \cdots \cdot a}_{n \text{ times}}}$$

where $a > 0$ is real.

Generalizing this definition involves first noting its abstract mathematical properties, and then making sure these properties are preserved in the generalization.

3.1.2 Properties of Exponents

From the basic definition of positive integer exponents, we have

(1) $a^{n_1}a^{n_2} = a^{n_1+n_2}$

(2) $(a^{n_1})^{n_2} = a^{n_1 n_2}$

Note that property (1) implies property (2). We list them both explicitly for convenience below.

3.1.3 The Exponent Zero

How should we define a^0 in a manner that is consistent with the properties of integer exponents? Multiplying it by a gives

$$a^0 \cdot a = a^0 a^1 = a^{0+1} = a^1 = a$$

by property (1) of exponents. Solving $a^0 \cdot a = a$ for a^0 then gives

$$\boxed{a^0 = 1}.$$

3.1.4 Negative Exponents

What should a^{-1} be? Multiplying it by a gives

$$a^{-1} \cdot a = a^{-1}a^1 = a^{-1+1} = a^0 = 1$$

Solving $a^{-1} \cdot a = 1$ for a^{-1} then gives

$$\boxed{a^{-1} = \frac{1}{a}}$$

Similarly, we obtain

$$\boxed{a^{-M} = \frac{1}{a^M}}$$

for all integer values of M, i.e., $\forall M \in \mathcal{Z}$.

3.1.5 Rational Exponents

A *rational* number is a real number that can be expressed as a ratio of two *integers*:

$$x = \frac{L}{M}, \quad L \in \mathcal{Z},\ M \in \mathcal{Z}$$

Applying property (2) of exponents, we have

$$a^x = a^{L/M} = \left(a^{\frac{1}{M}}\right)^L$$

Thus, the only thing new is $a^{1/M}$. Since

$$\left(a^{\frac{1}{M}}\right)^M = a^{\frac{M}{M}} = a$$

we see that $a^{1/M}$ is the Mth root of a. This is sometimes written

$$\boxed{a^{\frac{1}{M}} \triangleq \sqrt[M]{a}}$$

The Mth root of a real (or complex) number is not unique. As we all know, square roots give two values (e.g., $\sqrt{4} = \pm 2$). In the general case of Mth roots, there are M distinct values, in general.

How do we come up with M *different* numbers which when raised to the Mth power will yield a? The answer is to consider *complex* numbers in *polar form*. By Euler's Identity, the real number $a > 0$ can be expressed, for any integer k, as $a \cdot e^{j2\pi k} = a \cdot \cos(2\pi k) + j \cdot a \cdot \sin(2\pi k) = a + j \cdot a \cdot 0 = a$. Using this form for a, the number $a^{1/M}$ can be written as

$$\boxed{a^{\frac{1}{M}} = a^{\frac{1}{M}} e^{j2\pi k/M}, \qquad k = 0, 1, 2, 3, \ldots, M-1}$$

We can now see that we get a *different complex number* for each $k = 0, 1, 2, 3, \ldots, M-1$. When $k = M$, we get the same thing as when $k = 0$. When $k = M + 1$, we get the same thing as when $k = 1$, and so on, so there are only M cases using this construct.

Roots of Unity

When $a = 1$, we can write

$$1^{k/M} = e^{j2\pi k/M}, \quad k = 0, 1, 2, 3, \ldots, M-1$$

The special case $k = 1$ is called the *primitive Mth root of unity*, since integer powers of it give all of the others:

$$e^{j2\pi k/M} = \left(e^{j2\pi/M}\right)^k$$

The Mth roots of unity are so important that they are often given a special notation in the signal processing literature:

$$W_M^k \stackrel{\Delta}{=} e^{j2\pi k/M}, \qquad k = 0, 1, 2, \ldots, M-1,$$

where W_M denotes the *primitive* Mth root of unity. We may also call W_M the *generator* of the mathematical *group* consisting of the Mth roots of unity and their products.

We will learn later that the Nth roots of unity are used to generate all the sinusoids used by the DFT and its inverse. The kth sinusoid is given by

$$W_N^{kn} = e^{j2\pi kn/N} \stackrel{\Delta}{=} e^{j\omega_k t_n} = \cos(\omega_k t_n) + j\sin(\omega_k t_n), \quad n = 0, 1, 2, \ldots, N-1$$

where $\omega_k \stackrel{\Delta}{=} 2\pi k/NT$, $t_n \stackrel{\Delta}{=} nT$, and T is the sampling interval in seconds.

3.1.6 Real Exponents

The closest we can actually get to most real numbers is to compute a *rational number* that is as close as we need. It can be shown that rational numbers are *dense* in the real numbers; that is, between every two real numbers there is a rational number, and between every two rational numbers is a real number. An *irrational* number can be defined as any real number having a non-repeating decimal expansion. For example, $\sqrt{2}$ is an irrational real number whose decimal expansion starts out as

$$\sqrt{2} = 1.41421356237309504880168872420969807856967187537694807317667973 7\ldots$$

(computed via `N[Sqrt[2],80]` in Mathematica). Every truncated, rounded, or repeating expansion is a *rational* number. That is, it can be rewritten as an integer divided by another integer. For example,

$$1.414 = \frac{1414}{1000}$$

and, using $\overline{\text{overbar}}$ to denote the repeating part of a decimal expansion,

$$\begin{aligned} x &= 0.\overline{123} \\ \Rightarrow \quad 1000x &= 123.\overline{123} = 123 + x \\ \Rightarrow \quad 999x &= 123 \\ \Rightarrow \quad x &= \frac{123}{999} \end{aligned}$$

Other examples of irrational numbers include

$$\begin{aligned} \pi &= 3.14159265358979323846264338327950288419716939937510582097 49\ldots \\ e &= 2.71828182845904523536028747135266249775724709369995957496 69\ldots \end{aligned}$$

Let $\hat{x}_n$ denote the n-digit decimal expansion of an arbitrary real number x. Then $\hat{x}_n$ is a rational number (some integer over 10^n). We can say

$$\lim_{n\to\infty} \hat{x}_n = x$$

Since $a^{\hat{x}_n}$ is defined for all n, it is straightforward to define

$$a^x \stackrel{\Delta}{=} \lim_{n\to\infty} a^{\hat{x}_n}$$

3.1.7 A First Look at Taylor Series

Any "smooth" function $f(x)$ can be expanded in the form of a *Taylor series*:

$$f(x) = f(x_0) + \frac{f'(x_0)}{1}(x-x_0) + \frac{f''(x_0)}{1\cdot 2}(x-x_0)^2 + \frac{f'''(x_0)}{1\cdot 2\cdot 3}(x-x_0)^3 + \cdots .$$

This can be written more compactly as

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n.$$

An informal derivation of this formula for $x_0 = 0$ is given in §3.2 and §3.3. Clearly, since many derivatives are involved, a Taylor series expansion is only possible when the function is so smooth that it can be differentiated again and again. Fortunately for us, all audio signals can be defined so

as to be in that category. This is because hearing is bandlimited to 20 kHz, and any sum of sinusoids up to some maximum frequency, i.e., any audible signal, is infinitely differentiable. (Recall that $\sin'(x) = \cos(x)$ and $\cos'(x) = -\sin(x)$, etc.). See §3.6 for more on this topic.

3.1.8 Imaginary Exponents

We may define *imaginary exponents* the same way that all sufficiently smooth real-valued functions of a real variable are generalized to the complex case—using *Taylor series*. A Taylor series expansion is just a polynomial (possibly of infinitely high order), and polynomials involve only addition, multiplication, and division. Since these elementary operations are also defined for complex numbers, any smooth function of a real variable $f(x)$ may be generalized to a function of a complex variable $f(z)$ by simply substituting the complex variable $z = x + jy$ for the real variable x in the Taylor series expansion.

Let $f(x) \stackrel{\Delta}{=} a^x$, where a is any positive real number. The Taylor series expansion expansion about $x_0 = 0$ ("Maclaurin series"), generalized to the complex case is then

$$a^z \stackrel{\Delta}{=} f(0) + f'(0)(z) + \frac{f''(0)}{2}z^2 + \frac{f'''(0)z^3}{3!} + \cdots\cdots$$

which is well defined (although we should make sure the series *converges* for every finite z). We have $f(0) \stackrel{\Delta}{=} a^0 = 1$, so the first term is no problem. But what is $f'(0)$? In other words, what is the derivative of a^x at $x = 0$? Once we find the successive derivatives of $f(x) \stackrel{\Delta}{=} a^x$ at $x = 0$, we will be done with the definition of a^z for any complex z.

3.1.9 Derivatives of $f(x) = a^x$

Let's apply the definition of differentiation and see what happens:

$$\begin{aligned} f'(x_0) &\stackrel{\Delta}{=} \lim_{\delta\to 0}\frac{f(x_0+\delta)-f(x_0)}{\delta} \\ &\stackrel{\Delta}{=} \lim_{\delta\to 0}\frac{a^{x_0+\delta}-a^{x_0}}{\delta} = \lim_{\delta\to 0} a^{x_0}\frac{a^\delta - 1}{\delta} = a^{x_0}\lim_{\delta\to 0}\frac{a^\delta-1}{\delta} \end{aligned}$$

Since the limit of $(a^\delta - 1)/\delta$ as $\delta \to 0$ is less than 1 for $a = 2$ and greater than 1 for $a = 3$ (as one can show via direct calculations), and since

$(a^\delta - 1)/\delta$ is a continuous function of a, it follows that there exists a positive real number we'll call e such that for $a = e$ we get

$$\lim_{\delta\to 0} \frac{e^\delta - 1}{\delta} \stackrel{\Delta}{=} 1.$$

For $a = e$, we thus have $(a^x)' = (e^x)' = e^x$.

So far we have proved that the derivative of e^x is e^x. What about a^x for other values of a? The trick is to write it as

$$a^x = e^{\ln(a^x)} = e^{x\ln(a)}$$

and use the chain rule, where $\ln(a) \stackrel{\Delta}{=} \log_e(a)$ denotes the log-base-e of a. Formally, the chain rule tells us how do differentiate a function of a function as follows:

$$\frac{d}{dx} f(g(x))|_{x=x_0} = f'(g(x_0))g'(x_0)$$

In this case, $g(x) = x\ln(a)$ so that $g'(x) = \ln(a)$, and $f(y) = e^y$ which is its own derivative. The end result is then $(a^x)' = \left(e^{x\ln a}\right)' = e^{x\ln(a)}\ln(a) = a^x\ln(a)$, i.e.,

$$\boxed{\frac{d}{dx}a^x = a^x\ln(a)}$$

3.1.10 Back to e

Above, we defined e as the particular real number satisfying

$$\lim_{\delta\to 0} \frac{e^\delta - 1}{\delta} \stackrel{\Delta}{=} 1.$$

which gave us $(a^x)' = a^x$ when $a = e$. From this expression, we have, as $\delta \to 0$,

$$\begin{aligned} e^\delta - 1 &\to \delta \\ \Rightarrow \quad e^\delta &\to 1 + \delta \\ \Rightarrow \quad e &\to (1+\delta)^{1/\delta}, \end{aligned}$$

or,

$$\boxed{e \stackrel{\Delta}{=} \lim_{\delta\to 0}(1+\delta)^{1/\delta}}$$

This is one way to define e. Another way to arrive at the same definition is to ask what logarithmic base e gives that the derivative of $\log_e(x)$ is $1/x$. We denote $\log_e(x)$ by $\ln(x)$.

3.1.11 Sidebar on Mathematica

Mathematica is a handy tool for cranking out any number of digits in transcendental numbers such as e:

```
In[]:
  N[E,50]
Out[]:
  2.7182818284590452353602874713526624977572470937
```

Alternatively, we can compute $(1+\delta)^{1/\delta}$ for small δ:

```
In[]:
  (1+delta)^(1/delta) /. delta->0.001
Out[]:
  2.716923932235594
In[]:
  (1+delta)^(1/delta) /. delta->0.0001
Out[]:
  2.718145926824926
In[]:
  (1+delta)^(1/delta) /. delta->0.00001
Out[]:
  2.718268237192297
```

What happens if we just go for it and set delta to zero?

```
In[]:
  (1+delta)^(1/delta) /. delta->0
                                  1
Power::infy: Infinite expression - encountered.
                                  0
Infinity::indt:
                              ComplexInfinity
   Indeterminate expression 1                encountered.
Indeterminate
```

3.1.12 Back to $e^{j\theta}$

We've now defined a^z for any positive real number a and any complex number z. Setting $a = e$ and $z = j\theta$ gives us the special case we need for

Euler's identity. Since e^z is its own derivative, the Taylor series expansion for for $f(x) = e^x$ is one of the simplest imaginable infinite series:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots$$

The simplicity comes about because $f^{(n)}(0) = 1$ for all n and because we chose to expand about the point $x = 0$. We of course define

$$e^{j\theta} \triangleq \sum_{n=0}^{\infty} \frac{(j\theta)^n}{n!} = 1 + j\theta - \theta^2/2 - j\theta^3/3! + \cdots$$

Note that all even order terms are real while all odd order terms are imaginary. Separating out the real and imaginary parts gives

$$\begin{aligned} \mathrm{re}\left\{e^{j\theta}\right\} &= 1 - \theta^2/2 + \theta^4/4! - \cdots \\ \mathrm{im}\left\{e^{j\theta}\right\} &= \theta - \theta^3/3! + \theta^5/5! - \cdots \end{aligned}$$

Comparing the Maclaurin expansion for $e^{j\theta}$ with that of $\cos(\theta)$ and $\sin(\theta)$ proves Euler's identity. Recall that

$$\begin{aligned} \frac{d}{d\theta}\cos(\theta) &= -\sin(\theta) \\ \frac{d}{d\theta}\sin(\theta) &= \cos(\theta) \end{aligned}$$

so that

$$\begin{aligned} \left.\frac{d^n}{d\theta^n}\cos(\theta)\right|_{\theta=0} &= \begin{cases} (-1)^{n/2}, & n \text{ even} \\ 0, & n \text{ odd} \end{cases} \\ \left.\frac{d^n}{d\theta^n}\sin(\theta)\right|_{\theta=0} &= \begin{cases} (-1)^{(n-1)/2}, & n \text{ odd} \\ 0, & n \text{ even} \end{cases} \end{aligned}$$

Plugging into the general Maclaurin series gives

$$\begin{aligned} \cos(\theta) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!}\theta^n \\ &= \sum_{\substack{n\geq 0 \\ n \text{ even}}}^{\infty} \frac{(-1)^{n/2}}{n!}\theta^n \\ \sin(\theta) &= \sum_{\substack{n\geq 0 \\ n \text{ odd}}}^{\infty} \frac{(-1)^{(n-1)/2}}{n!}\theta^n \end{aligned}$$

Separating the Maclaurin expansion for $e^{j\theta}$ into its even and odd terms (real and imaginary parts) gives

$$e^{j\theta} \triangleq \sum_{n=0}^{\infty} \frac{(j\theta)^n}{n!} = \sum_{\substack{n\geq 0 \\ n \text{ even}}}^{\infty} \frac{(-1)^{n/2}}{n!}\theta^n + j \sum_{\substack{n\geq 0 \\ n \text{ odd}}}^{\infty} \frac{(-1)^{(n-1)/2}}{n!}\theta^n = \cos(\theta) + j\sin(\theta)$$

thus proving Euler's identity.

3.2 Informal Derivation of Taylor Series

We have a function $f(x)$ and we want to approximate it using an nth-order *polynomial*:

$$f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_n x^n + R_{n+1}(x)$$

where $R_{n+1}(x)$, which is obviously the approximation error, is called the "remainder term." We may assume x and $f(x)$ are *real*, but the following derivation generalizes unchanged to the complex case.

Our problem is to find fixed constants $\{f_i\}_{i=0}^n$ so as to obtain the best approximation possible. Let's proceed optimistically as though the approximation will be perfect, and assume $R_{n+1}(x) = 0$ for all x ($R_{n+1}(x) \equiv 0$), given the right values of f_i. Then at $x = 0$ we must have

$$f(0) = f_0$$

That's one constant down and $n-1$ to go! Now let's look at the first derivative of $f(x)$ with respect to x, again assuming that $R_{n+1}(x) \equiv 0$:

$$f'(x) = 0 + f_1 + 2f_2 x + 3f_2 x^2 + \cdots + nf_n x^{n-1}$$

Evaluating this at $x = 0$ gives

$$f'(0) = f_1$$

In the same way, we find

$$\begin{aligned} f''(0) &= 2 \cdot f_2 \\ f'''(0) &= 3 \cdot 2 \cdot f_3 \\ &\cdots \\ f^{(n)}(0) &= n! \cdot f_n \end{aligned}$$

where $f^{(n)}(0)$ denotes the nth derivative of $f(x)$ with respect to x, evaluated at $x = 0$. Solving the above relations for the desired constants yields

$$\begin{aligned} f_0 &= f(0) \\ f_1 &= \frac{f'(0)}{1} \\ f_2 &= \frac{f''(0)}{2 \cdot 1} \\ f_3 &= \frac{f'''(0)}{3 \cdot 2 \cdot 1} \\ &\cdots \\ f_n &= \frac{f^{(n)}(0)}{n!} \end{aligned}$$

Thus, defining $0! \stackrel{\Delta}{=} 1$ (as it always is), we have derived the following polynomial approximation:

$$\boxed{f(x) \approx \sum_{k=0}^{n} \frac{f^{(k)}(0)}{k!} x^k}$$

This is the nth-order Taylor series expansion of $f(x)$ about the point $x = 0$. Its derivation was quite simple. The hard part is showing that the approximation error (remainder term $R_{n+1}(x)$) is small over a wide interval of x values. Another "math job" is to determine the conditions under which the approximation error approaches zero for all x as the order n goes to infinity. The main point to note here is that the form of the Taylor series expansion itself is simple to derive.

3.3 Taylor Series with Remainder

We repeat the derivation of the preceding section, but this time we treat the error term more carefully.

Again we want to approximate $f(x)$ with an nth-order *polynomial*:

$$f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_n x^n + R_{n+1}(x)$$

$R_{n+1}(x)$ is the "remainder term" which we will no longer assume is zero.

Our problem is to find $\{f_i\}_{i=0}^{n}$ so as to minimize $R_{n+1}(x)$ over some interval I containing x. There are many "optimality criteria" we could

choose. The one that falls out naturally here is called "Padé" approximation. Padé approximation sets the error value and its first n derivatives to zero at a single chosen point, which we take to be $x = 0$. Since all $n + 1$ "degrees of freedom" in the polynomial coefficients f_i are used to set derivatives to zero at one point, the approximation is termed "maximally flat" at that point. Padé approximation comes up often in signal processing. For example, it is the sense in which Butterworth lowpass filters are optimal. (Their frequency reponses are maximally flat at dc.) Also, Lagrange interpolation filters can be shown to maximally flat at dc in the frequency domain.

Setting $x = 0$ in the above polynomial approximation produces

$$f(0) = f_0 + R_{n+1}(0) = f_0$$

where we have used the fact that the error is to be exactly zero at $x = 0$.

Differentiating the polynomial approximation and setting $x = 0$ gives

$$f'(0) = f_1 + R'_{n+1}(0) = f_1$$

where we have used the fact that we also want the *slope* of the error to be exactly zero at $x = 0$.

In the same way, we find

$$f^{(k)}(0) = k! \cdot f_k + R^{(k)}_{n+1}(0) = k! \cdot f_k$$

for $k = 2, 3, 4, \ldots, n$, and the first n derivatives of the remainder term are all zero. Solving these relations for the desired constants yields the nth-order Taylor series expansion of $f(x)$ about the point $x = 0$

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(0)}{k!} x^k + R_{n+1}(x)$$

as before, but now we better understand the remainder term.

From this derivation, it is clear that the approximation error (remainder term) is smallest in the vicinity of $x = 0$. *All degrees of freedom* in the polynomial coefficients were devoted to minimizing the approximation error and its derivatives at $x = 0$. As you might expect, the approximation error generally worsens as x gets farther away from 0.

To obtain a more *uniform* approximation over some interval I in x, other kinds of error criteria may be employed. This is classically called "economization of series," but nowadays we may simply call it *polynomial*

approximation under different error criteria. In `Matlab`, the function `polyfit(x,y,n)` will find the coefficients of a polynomial $p(x)$ of degree `n` that fits the data `y` over the points `x` in a least-squares sense. That is, it minimizes

$$\| R_{n+1} \|^2 \triangleq \sum_{i=1}^{n_x} |y(i) - p(x(i))|^2$$

where $n_x \triangleq$ `length(x)`.

3.4 Formal Statement of Taylor's Theorem

Let $f(x)$ be continuous on a real interval I containing x_0 (and x), and let $f^{(n)}(x)$ exist at x and $f^{(n+1)}(\xi)$ be continuous for all $\xi \in I$. Then we have the following Taylor series expansion:

$$\begin{aligned} f(x) = f(x_0) &+ \frac{1}{1}f'(x_0)(x-x_0) \\ &+ \frac{1}{1\cdot 2}f''(x_0)(x-x_0)^2 \\ &+ \frac{1}{1\cdot 2\cdot 3}f'''(x_0)(x-x_0)^3 \\ &+ \cdots \\ &+ \frac{1}{n!}f^{(n+1)}(x_0)(x-x_0)^n \\ &+ R_{n+1}(x) \end{aligned}$$

where $R_{n+1}(x)$ is called the remainder term. There exists ξ between x and x_0 such that

$$R_{n+1}(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1}$$

In particular, if $|f^{(n+1)}| \leq M$ in I, then

$$R_{n+1}(x) \leq \frac{M|x-x_0|^{n+1}}{(n+1)!}$$

which is normally small when x is close to x_0.

When $x_0 = 0$, the Taylor series reduces to what is called a Maclaurin series.

3.5 Weierstrass Approximation Theorem

Let $f(x)$ be continuous on a real interval I. Then for any $\epsilon > 0$, there exists an nth-order polynomial $P_n(f,x)$, where n depends on ϵ, such that

$$|P_n(f,x) - f(x)| < \epsilon$$

for all $x \in I$.

Thus, any continuous function can be approximated arbitrarily well by means of a polynomial. Furthermore, an infinite-order polynomial can yield an error-free approximation. Of course, to compute the polynomial coefficients using a Taylor series expansion, the function must also be differentiable of all orders throughout I.

3.6 Differentiability of Audio Signals

As mentioned earlier, every audio signal can be regarded as infinitely differentiable due to the finite bandwidth of human hearing. One of the Fourier properties we will learn later in this reader is that *a signal cannot be both time limited and frequency limited.* Therefore, by conceptually "lowpass filtering" every audio signal to reject all frequencies above 20 kHz, we implicitly make every audio signal last forever! Another way of saying this is that the "ideal lowpass filter 'rings' forever". Such fine points do not concern us in practice, but they are important for fully understanding the underlying theory. Since, in reality, signals can be said to have a true beginning and end, we must admit in practice that all signals we work with have infinite-bandwidth at turn-on and turn-off transients.[1]

[1] One joke along these lines, due, I'm told, to Professor Bracewell, is that "since the telephone is bandlimited to 3kHz, and since bandlimited signals cannot be time limited, it follows that one cannot hang up the telephone".

Chapter 4

Logarithms, Decibels, and Number Systems

This chapter provides an introduction to logarithms (real and complex), decibels, and number systems such as binary integer fixed-point, fractional fixed-point, one's complement, two's complement, logarithmic fixed-point, μ-law, and floating-point number formats.

4.1 Logarithms

A *logarithm* $y = \log_b(x)$ is fundamentally an *exponent* y applied to a specific *base* b. That is, $x = b^y$. The term "logarithm" can be abbreviated as "log". The base b is chosen to be a positive real number, and we normally only take logs of positive real numbers $x > 0$ (although it is ok to say that the log of 0 is $-\infty$). The inverse of a logarithm is called an *antilogarithm* or *antilog*.

For any positive number x, we have

$$x = b^{\log_b(x)}$$

for any valid base $b > 0$. This is just an identity arising from the definition of the logarithm, but it is sometimes useful in manipulating formulas.

When the base is not specified, it is normally assumed to be 10, i.e., $\log(x) \stackrel{\Delta}{=} \log_{10}(x)$. This is the *common logarithm*.

Base 2 and base e logarithms have their own special notation:

$$\begin{aligned} \ln(x) &\stackrel{\Delta}{=} \log_e(x) \\ \lg(x) &\stackrel{\Delta}{=} \log_2(x) \end{aligned}$$

(The use of lg() for base 2 logarithms is common in computer science. In mathematics, it may denote a base 10 logarithm.) By far the most common bases are 10, e, and 2. Logs base e are called *natural logarithm*s. They are "natural" in the sense that

$$\frac{d}{dx}\ln(x) = \frac{1}{x}$$

while the derivatives of logarithms to other bases are not quite so simple:

$$\frac{d}{dx}\log_b(x) = \frac{1}{x\ln(b)}$$

(Prove this as an exercise.) The inverse of the natural logarithm $y = \ln(x)$ is of course the exponential function $x = e^y$, and e^y is its own derivative.

In general, a logarithm y has an integer part and a fractional part. The integer part is called the *characteristic of the logarithm*, and the fractional part is called the *mantissa*. These terms were suggested by Henry Briggs in 1624. "Mantissa" is a Latin word meaning "addition" or "make weight"—something added to make up the weight [3].

The following Matlab code illustrates splitting a natural logarithm into its characteristic and mantissa:

```
>> x = log(3)
   x = 1.0986
>> characteristic = floor(x)
   characteristic = 1
>> mantissa = x - characteristic
   mantissa = 0.0986

>> % Now do a negative-log example
>> x = log(0.05)
   x = -2.9957
>> characteristic = floor(x)
   characteristic = -3
>> mantissa = x - characteristic
   mantissa = 0.0043
```

Logarithms were used in the days before computers to perform *multiplication of large numbers*. Since $\log(xy) = \log(x) + \log(y)$, one can look up the logs of x and y in tables of logarithms, add them together (which is easier than multiplying), and look up the antilog of the result to obtain

the product xy. Log tables are still used in modern computing environments to replace expensive multiplies with less-expensive table lookups and additions. This is a classic tradeoff between memory (for the log tables) and computation. Nowadays, large numbers are multiplied using FFT fast-convolution techniques.

4.1.1 Changing the Base

By definition, $x = b^{\log_b(x)}$. Taking the log base a of both sides gives

$$\log_a(x) = \log_b(x)\log_a(b)$$

which tells how to convert the base from b to a, that is, how to convert the log base b of x to the log base a of x. (Just multiply by the log base a of b.)

4.1.2 Logarithms of Negative and Imaginary Numbers

By *Euler's formula*, $e^{j\pi} = -1$, so that

$$\ln(-1) = j\pi$$

from which it follows that for any $x < 0$, $\ln(x) = j\pi + \ln(|x|)$.

Similarly, $e^{j\pi/2} = j$, so that

$$\ln(j) = j\frac{\pi}{2}$$

and for any imaginary number $z = jy$, $\ln(z) = j\pi/2 + \ln(y)$, where y is real.

Finally, from the polar representation $z = re^{j\theta}$ for complex numbers,

$$\ln(z) \stackrel{\Delta}{=} \ln(re^{j\theta}) = \ln(r) + j\theta$$

where $r > 0$ and θ are real. Thus, the log of the magnitude of a complex number behaves like the log of any positive real number, while the log of its phase term $e^{j\theta}$ extracts its phase (times j).

As an example of the use of logarithms in signal processing, note that the negative imaginary part of the derivative of the log of a *spectrum*

$X(\omega)$ is defined as the *group delay*[1] of the signal $x(t)$:

$$D_x(\omega) \stackrel{\Delta}{=} -\mathrm{im}\left\{\frac{d}{d\omega}\ln(X(\omega))\right\}$$

Another usage is in *Homomorphic Signal Processing* [6, Chapter 10] in which the multiplicative formants in vocal spectra are converted to additive low-frequency variations in the spectrum (with the harmonics being the high-frequency variation in the spectrum). Thus, the lowpass-filtered log spectrum contains only the formants, and the complementarily highpass-filtered log spectrum contains only the fine structure associated with the pitch.

> **Exercise:** Work out the definition of logarithms using a *complex* base b.

4.2 Decibels

A *decibel* (abbreviated dB) is defined as one tenth of a *bel*. The bel[2] is an amplitude unit defined for sound as the log (base 10) of the *intensity* relative to some *reference intensity*,[3] i.e.,

$$\text{Amplitude\_in\_bels} = \log_{10}\left(\frac{\text{Signal\_Intensity}}{\text{Reference\_Intensity}}\right)$$

The choice of reference intensity (or power) defines the particular choice of *dB scale*. Signal intensity, power, and energy are always proportional to the *square* of the signal *amplitude*. Thus, we can always translate these

[1] Group delay and phase delay are covered in the CCRMA publication [4] as well as in standard signal processing references [5]. In the case of an AM or FM broadcast signal which is passed through a filter, the *carrier wave* is delayed by the *phase delay* of the filter, while the *modulation signal* is delayed by the *group delay* of the filter. In the case of additive synthesis, group delay applies to the amplitude envelope of each sinusoidal oscillator, while the phase delay applies to the sinusoidal oscillator waveform itself.

[2] The "bel" is named after Alexander Graham Bell, the inventor of the telephone.

[3] *Intensity* is physically *power per unit area*. Bels may also be defined in terms of *energy*, or *power* which is energy per unit time. Since sound is always measured over some *area* by a microphone diaphragm, its physical power is conventionally normalized by area, giving intensity. Similarly, the *force* applied by sound to a microphone diaphragm is normalized by area to give *pressure* (force per unit area).

energy-related measures into squared amplitude:

$$\text{Amplitude\_in\_bels} = \log_{10}\left(\frac{\text{Amplitude}^2}{\text{Amplitude}^2_{\text{ref}}}\right) = 2\log_{10}\left(\frac{|\text{Amplitude}|}{|\text{Amplitude}_{\text{ref}}|}\right)$$

Since there are 10 decibels to a bel, we also have

$$\begin{aligned}\text{Amplitude}_{\text{dB}} &= 20\log_{10}\left(\frac{|\text{Amplitude}|}{|\text{Amplitude}_{\text{ref}}|}\right) = 10\log_{10}\left(\frac{\text{Intensity}}{\text{Intensity}_{\text{ref}}}\right)\\ &= 10\log_{10}\left(\frac{\text{Power}}{\text{Power}_{\text{ref}}}\right) = 10\log_{10}\left(\frac{\text{Energy}}{\text{Energy}_{\text{ref}}}\right)\end{aligned}$$

A *just-noticeable difference* (JND) in amplitude level is on the order of a quarter dB. In the early days of telephony, one dB was considered a reasonable "smallest step" in amplitude, but in reality, a series of half-dB amplitude steps does not sound very smooth, while quarter-dB steps do sound pretty smooth. A typical professional audio filter-design specification for "ripple in the passband" is 0.1 dB.

> **Exercise:** Try synthesizing a sawtooth waveform which increases by 1/2 dB a few times per second, and again using 1/4 dB increments. See if you agree that quarter-dB increments are "smooth" enough for you.

4.2.1 Properties of DB Scales

In every kind of dB, a *factor of 10* in amplitude *gain* corresponds to a *20 dB boost* (increase by 20 dB):

$$20\log_{10}\left(\frac{10\cdot A}{A_{\text{ref}}}\right) = \underbrace{20\log_{10}(10)}_{20\text{ dB}} + 20\log_{10}\left(\frac{A}{A_{\text{ref}}}\right)$$

and $20\log_{10}(10) = 20$, of course. A function $f(x)$ which is proportional to $1/x$ is said to "fall off" (or "roll off") at the rate of 20 *dB per decade*. That is, for every factor of 10 in x (every "decade"), the amplitude drops 20 dB.

Similarly, a factor of 2 in amplitude gain corresponds to a 6 dB boost:

$$20\log_{10}\left(\frac{2\cdot A}{A_{\text{ref}}}\right) = \underbrace{20\log_{10}(2)}_{6\text{ dB}} + 20\log_{10}\left(\frac{A}{A_{\text{ref}}}\right)$$

and

$$20\log_{10}(2) = 6.0205999\ldots \approx 6 \text{ dB}.$$

A function $f(x)$ which is proportional to $1/x$ is said to fall off 6 *dB per octave*. That is, for every factor of 2 in x (every "octave"), the amplitude drops close to 6 dB. Thus, 6 dB per octave is the same thing as 20 dB per decade.

A *doubling of power* corresponds to a *3 dB boost*:

$$10\log_{10}\left(\frac{2\cdot A^2}{A_{\text{ref}}^2}\right) = \underbrace{10\log_{10}(2)}_{3 \text{ dB}} + 10\log_{10}\left(\frac{A^2}{A_{\text{ref}}^2}\right)$$

and

$$10\log_{10}(2) = 3.010\ldots \approx 3 \text{ dB}.$$

Finally, note that the choice of *reference* merely determines a vertical offset in the dB scale:

$$20\log_{10}\left(\frac{A}{A_{\text{ref}}}\right) = 20\log_{10}(A) - \underbrace{20\log_{10}(A_{\text{ref}})}_{\text{constant offset}}$$

4.2.2 Specific DB Scales

Since we so often rescale our signals to suit various needs (avoiding overflow, reducing quantization noise, making a nicer plot, etc.), there seems to be little point in worrying about what the dB reference is—we simply choose it implicitly when we rescale to obtain signal values in the range we want to see. Nevertheless, a few specific dB scales are worth knowing about.

DBm Scale

One common dB scale in audio recording is the dBm scale in which the reference power is taken to be a milliwatt (1 mW) dissipated by a 600 Ohm resistor. (See Appendix 4.6 for a primer on resistors, voltage, current, and power.)

DBV Scale

Another dB scale is the dBV scale which sets 0 dBV to 1 volt. Thus, a 100-volt signal is

$$20\log_{10}\left(\frac{100V}{1V}\right) = 40 \text{ dBV}$$

and a 1000-volt signal is

$$20\log_{10}\left(\frac{1000V}{1V}\right) = 60 \text{ dBV}$$

Note that the dBV scale is undefined for current or power, unless the voltage is assumed to be across a standard resistor value, such as 600 Ohms.

DB SPL

Sound Pressure Level (SPL) is defined using a reference which is approximately the intensity of 1000 Hz sinusoid that is just barely audible (0 "phons"). In *pressure* units:

$$\begin{aligned}
0 \text{ dB SPL} &\triangleq 0.0002\ \mu\text{bar (micro-barometric pressure}[4]) \\
&= 20\ \mu\text{Pa (micro-Pascals)} \\
&= 2.9\times 10^{-9}\ \text{PSI (pounds per square inch)} \\
&= 2\times 10^{-4}\,\frac{\text{dynes}}{\text{cm}^2} \quad \text{(CGS units)} \\
&= 2\times 10^{-5}\,\frac{\text{nt}}{\text{m}^2} \quad \text{(MKS units)}
\end{aligned}$$

In *intensity* units:

$$I_0 = 10^{-16}\frac{\text{W}}{\text{cm}^2}$$

which corresponds to a root-mean-square (rms) pressure amplitude of 20.4 μPa, or about 20 μPa, as listed above. The *wave impedance* of air plays the role of "resistor" in relating the pressure- and intensity-based references exactly analogous to the dBm case discussed above.

Since *sound* is created by a time-varying pressure, we compute sound levels in dB-SPL by using the *average* intensity (averaged over at least one period of the lowest frequency contained in the sound).

Table 4.1 gives a list list of common sound levels and their dB equivalents[5] [7]: In my experience, the "threshold of pain" is most often defined as 120 dB.

The relationship between sound amplitude and actual *loudness* is complex [8]. Loudness is a perceptual dimension while sound amplitude is

[5] Adapted from S. S. Stevens, F. Warshofsky, and the Editors of Time-Life Books, *Sound and Hearing*, Life Science Library, Time-Life Books, Alexandria, VA, 1965, p. 173.

Sound	**dB-SPL**
Jet engine at 3m	140
Threshold of pain	130
Rock concert	120
Accelerating motorcycle at 5m	110
Pneumatic hammer at 2m	100
Noisy factory	90
Vacuum cleaner	80
Busy traffic	70
Quiet restaurant	50
Residential area at night	40
Empty movie house	30
Rustling of leaves	20
Human breathing (at 3m)	10
Threshold of hearing (good ears)	0

Table 4.1: Ballpark figures for the dB-SPL level of common sounds.

physical. Since loudness sensitivity is closer to logarithmic than linear in amplitude (especially at moderate to high loudnesses), we typically use decibels to represent sound amplitude, especially in spectral displays.

The *sone amplitude scale* is defined in terms of actual loudness perception experiments [8]. At 1kHz and above, loudness perception is approximately logarithmic above 50 dB SPL or so. Below that, it tends toward being more linear.

The *phon amplitude scale* is simply the dB scale at 1kHz [8, p. 111]. At other frequencies, the amplitude in phons is defined by following the equal-loudness curve over to 1 kHz and reading off the level there in dB SPL. In other words, all pure tones have the same loudness at the same phon level, and 1 kHz is used to set the reference in dB SPL. Just remember that one phon is one dB-SPL at 1 kHz. Looking at the Fletcher-Munson equal-loudness curves [8, p. 124], loudness in phons can be read off along the vertical line at 1 kHz.

Classically, the *intensity level* of a sound wave is its dB SPL level, measuring the peak time-domain pressure-wave amplitude relative to 10^{-16} watts per centimeter squared (i.e., there is no consideration of the frequency domain here at all).

Another classical term still encountered is the *sensation level* of pure

tones: The sensation level is the number of dB SPL above the *hearing threshold* at that frequency [8, p. 110].

For further information on "doing it right," see, for example, http://www.measure.demon.co.uk/Acoustics_Software/loudness.html.

DB for Display

In practical signal processing, it is common to choose the *maximum signal magnitude* as the reference amplitude. That is, we normalize the signal so that the maximum amplitude is defined as 1, or 0 dB. This convention is also used by "sound level meters" in audio recording. When displaying magnitude spectra, the highest spectral peak is often normalized to 0 dB. We can then easily read off lower peaks as so many dB below the highest peak.

Figure 4.1b shows a plot of the Fast Fourier Transform (FFT) of ten periods of a "Kaiser-windowed" sinusoid at 440 Hz. (FFT windows will be discussed later in this reader. For now, just think of the window as selecting and tapering a finite-duration section of the signal.) Note that the peak dB magnitude has been normalized to zero, and that the plot has been clipped at -100 dB.

Below is the Matlab code for producing Fig. 4.1. Note that it contains several elements (windows, zero padding, spectral interpolation) that we will not cover until later. They are included here as "forward references" in order to keep the example realistic and practical, and to give you an idea of "how far we have to go" before we know how to do practical spectrum analysis. Otherwise, the example just illustrates plotting spectra on an arbitrary dB scale between convenient limits.

```
% Example practical display of the FFT of a synthesized sinusoid

fs = 44100;              % Sampling rate
f = 440;                 % Sinusoidal frequency = A-440
nper = 10;               % Number of periods to synthesize
dur = nper/f;            % Duration in seconds
T = 1/fs;                % Sampling period
t = 0:T:dur;             % Discrete-time axis in seconds
L = length(t)            % Number of samples to synthesize
ZP = 5;                  % Zero padding factor (for spectral interpolation)
N = 2^(nextpow2(L*ZP))   % FFT size (power of 2)
```

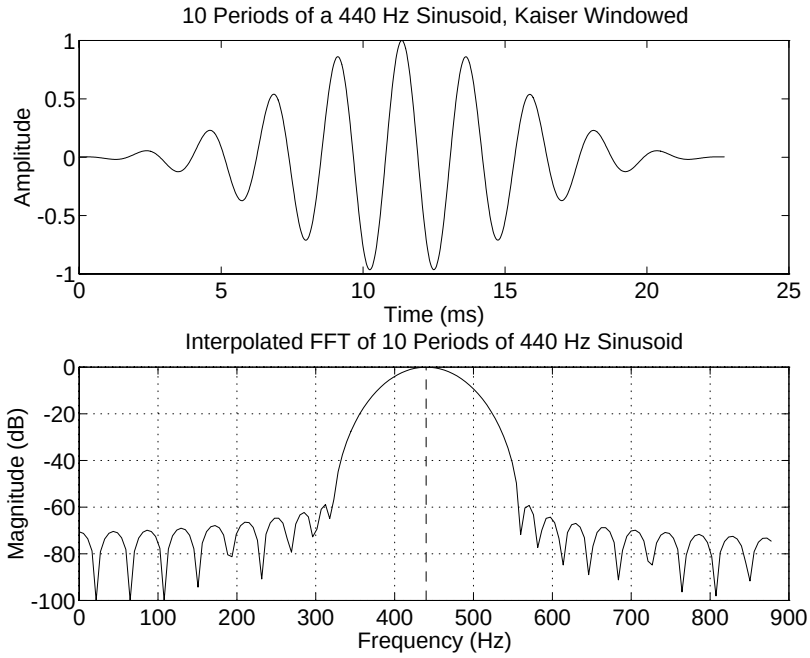

Figure 4.1: Windowed sinusoid (top) and its FFT magnitude (bottom).

```
x = cos(2*pi*f*t);      % A sinusoid at A-440 (a "row vector")
w = kaiser(L,8);        % We'll learn a bit about "FFT windows" later
xw = x .* w';           % Need to transpose w to get a row vector
sound(xw,fs);           % Might as well listen to it
xzp = [xw,zeros(1,N-L)];% Zero-padded FFT input buffer
X = fft(xzp);           % Spectrum of xw, interpolated by factor ZP

Xmag = abs(X);          % Spectral magnitude
Xdb = 20*log10(Xmag);   % Spectral magnitude in dB

XdbMax = max(Xdb);      % Peak dB magnitude
Xdbn = Xdb - XdbMax;    % Normalize to 0dB peak

dBmin = -100;           % Don't show anything lower than this
Xdbp = max(Xdbn,dBmin); % Normalized, clipped, dB magnitude spectrum
fmaxp = 2*f;            % Upper frequency limit of plot, in Hz
kmaxp = fmaxp*N/fs;     % Upper frequency limit of plot, in bins
fp = fs*[0:kmaxp]/N;    % Frequency axis in Hz

% Ok, plot it already!

subplot(2,1,1);
plot(1000*t,xw);
xlabel('Time (ms)');
ylabel('Amplitude');
title(sprintf('a) %d Periods of a %3.0f Hz Sinusoid, Kaiser Windowed',nper,f));

subplot(2,1,2);
plot(fp,Xdbp(1:kmaxp+1)); grid;
% Plot a dashed line where the peak should be:
  hold on; plot([440 440],[dBmin,0],'--'); hold off;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title(sprintf(['b) Interpolated FFT of %d Periods of ',...
'%3.0f Hz Sinusoid'],nper,f));
```

The following more compact Matlab produces essentially the same plot, but without the nice physical units on the horizontal axes:

```
x = cos([0:2*pi/20:10*2*pi]); % 10 periods of a sinusoid, 20 samples/cycle
L = length(x);
```

```
xw = x' .* kaiser(L,8);
N = 2^nextpow2(L*5);
X = fft([xw',zeros(1,N-L)]);

subplot(2,1,1); plot(xw);
xlabel('Time (samples)'); ylabel('Amplitude');
title('a) 10 Periods of a Kaiser-Windowed Sinusoid');

subplot(2,1,2); kmaxp = 2*10*5; Xl = 20*log10(abs(X(1:kmaxp+1)));
plot([10*5+1,10*5+1],[-100,0],[0:kmaxp],max(Xl - max(Xl),-100)); grid;
xlabel('Frequency (Bins)'); ylabel('Magnitude (dB)');
title('b) Interpolated FFT of 10 Periods of Sinusoid');
```

4.2.3 Dynamic Range

The *dynamic range* of a *signal processing system* can be defined as the maximum dB level sustainable without overflow (or other distortion) minus the dB level of the "noise floor".

Similarly, the dynamic range of a *signal* can be defined as its maximum decibel level minus its average "noise level" in dB. For digital signals, the limiting noise is ideally *quantization noise*.

Quantization noise is generally modeled as a uniform random variable between plus and minus half the least significant bit (since rounding to the nearest representable sample value is normally used). If q denotes the quantization interval, then the maximum quantization-error magnitude is $q/2$, and its variance ("noise power") is $\sigma_q^2 = q^2/12$ (see Appendix 4.5 for a derivation of this value).

The rms level of the quantization noise is therefore $\sigma_q = q/(2\sqrt{3}) \approx 0.3q$, or about 60% of the maximum error.

The *number system* and *number of bits* chosen to represent signal samples determines their available dynamic range. Signal processing operations such as digital filtering may use the same number system as the input signal, or they may use extra bits in the computations, yielding an increased "internal dynamic range".

Since the threshold of hearing is near 0 dB SPL, and since the "threshold of pain" is often defined as 120 dB SPL, we may say that the dynamic range of human hearing is approximately 120 dB.

The dynamic range of *magnetic tape* is approximately 55 dB. To increase the dynamic range available for analog recording on magnetic tape, *companding* is often used. "Dolby A" adds approximately 10 dB to the dynamic range that will fit on magnetic tape (by compressing the signal

dynamic range by 10 dB), while DBX adds 30 dB (at the cost of more "transient distortion").[6] In general, any dynamic range can be mapped to any other dynamic range, subject only to noise limitations.

4.3 Linear Number Systems for Digital Audio

This section discusses the most commonly used number formats for digital audio.

4.3.1 Pulse Code Modulation (PCM)

The "standard" number format for sampled audio signals is officially called *Pulse Code Modulation (*PCM*)*. This term simply means that each signal sample is interpreted as a "pulse" (e.g., a voltage or current pulse) at a particular amplitude which is binary encoded, typically in two's complement binary fixed-point format (discussed below). When someone says they are giving you a soundfile in "raw binary format", they pretty much always mean (nowadays) 16-bit, two's-complement PCM data. Most mainstream computer soundfile formats consist of a "header" (containing the length, etc.) followed by 16-bit two's-complement PCM.

You can normally convert a soundfile from one computer's format to another by stripping off its header and prepending the header for the new machine (or simply treating it as raw binary format on the destination computer). The UNIX "cat" command can be used for this, as can the Emacs text editor (which handles binary data just fine). The only issue usually is whether the bytes have to be swapped (an issue discussed further below).

4.3.2 Binary Integer Fixed-Point Numbers

Most prevalent computer languages only offer two kinds of numbers, *floating-point* and *integer fixed-point*. On present-day computers, all numbers are encoded using *binary digits* (called "bits") which are either 1 or 0.[7] In C, C++, and Java, floating-point variables are declared as `float`

[6] Companders (compresser-expanders) essentially "turn down" the signal gain when it is "loud" and "turn up" the gain when it is "quiet". As long as the input-output curve is monotonic (such as a log characteristic), the dynamic-range compression can be undone (expanded).

[7] Computers use bits, as opposed to the more familiar decimal digits, because they are more convenient to implement in digital hardware. For example, the *decimal num-*

(32 bits) or `double` (64 bits), while integer fixed-point variables are declared as `short int` (typically 16 bits and never less), `long int` (typically 32 bits and never less), or simply `int` (typically the same as a `long int`, but sometimes between short and long). For an 8-bit integer, one can use the `char` datatype (8 bits).

Since C was designed to accommodate a wide range of hardware, including old mini-computers, some lattitude was historically allowed in the choice of these bit-lengths. The `sizeof` operator is officially the "right way" for a C program to determine the number of bytes in various data types at run-time, e.g. `sizeof(long)`. (The word `int` can be omitted after `short` or `long`.) Nowadays, however, `short`s are always 16 bits (at least on all the major platforms), `int`s are 32 bits, and `long`s are typically 32 bits on 32-bit computers and 64 bits on 64-bit computers (although some C/C++ compilers use `long long int` to declare 64-bit ints). Table 4.2 gives the lengths currently used by GNU C/C++ compilers (usually called "`gcc`" or "`cc`") on 64-bit processors.[8]

Java, which is designed to be platform independent, defines a `long int` as *equivalent in precision to* 64 bits, an `int` as 32 bits, a `short int` as 16 bits, and additionally a `byte int` as an 8-bit int. Similarly, the "Structured Audio Orchestra Language" (SAOL[9]) (pronounced "sail")—the sound-synthesis component of the new MPEG-4 audio compression standard—requires only that the underlying number system be at least as accurate as 32-bit `float`s. All `int`s discussed thus far are *signed* integer formats. C and C++ also support *unsigned* versions of all `int` types, and they range from 0 to $2^N - 1$ instead of -2^{N-1} to $2^{N-1} - 1$, where N is

bers 0, 1, 2, 3, 4, 5 become, in binary format, 0, 1, 10, 11, 100, 101. Each *bit* position in binary notation corresponds to a power of 2, e.g., $5 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$; while each *digit* position in decimal notation corresponds to a power of 10, e.g., $123 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$. The term "digit" comes from the same word meaning "finger." Since we have ten fingers (digits), the term "digit" technically should be associated only with decimal notation, but in practice it is used for others as well. Other popular number systems in computers include *octal* which is base 8 (rarely seen any more, but still specifiable in any C/C++ program by using a leading zero, e.g., $0755 = 7 \cdot 8^2 + 5 \cdot 8^1 + 5 \cdot 8^0 = 493$ decimal = 111,101,101 binary), and *hexadecimal* (or simply "hex") which is base 16 and which employs the letters A through F to yield 16 digits (specifiable in C/C++ by starting the number with "0x", e.g., 0x1ED = $1 \cdot 16^2 + 15 \cdot 16^1 + 14 \cdot 16^0 = 493$ decimal = 1,1110,1101 binary). Note, however, that the representation within the computer is still always binary; octal and hex are simply convenient *groupings* of bits into sets of three bits (octal) or four bits (hex).

[8] This information is subject to change without notice. Check your local compiler documentation.

[9] http://sound.media.mit.edu/mpeg4

Type	Bytes	Notes
`char`	1	
`short`	2	
`int`	4	
`long`	8	(4 bytes on 32-bit machines)
`long long`	8	(may become 16 bytes)
`type *`	8	(any pointer)
`float`	4	
`double`	8	
`long double`	8	(may become 10 bytes)
`size_t`	8	(type of sizeof())
T* - T*	8	(pointer arithmetic)

Table 4.2: Byte sizes of GNU C/C++ data types for 64-bit architectures.

the number of bits. Finally, an `unsigned char` is often used for integers that only range between 0 and 255.

One's Complement Fixed-Point Format

One's Complement is a particular assignment of bit patterns to numbers. For example, in the case of 3-bit binary numbers, we have the assignments shown in Table 4.3.

Binary	Decimal
000	0
001	1
010	2
011	3
100	-3
101	-2
110	-1
111	-0

Table 4.3: Three-bit one's-complement binary fixed-point numbers.

In general, N-bit numbers are assigned to binary counter values in the "obvious way" as integers from 0 to $2^{N-1}-1$, and then the negative

numbers are assigned in reverse order, as shown in the example.

The term "one's complement" refers to the fact that negating a number in this format is accomplished by simply *complementing* the bit pattern (inverting each bit).

Note that there are two representations for zero (all 0s and all 1s). This is inconvenient when testing if a number is equal to zero. For this reason, one's complement is generally not used.

Two's Complement Fixed-Point Format

In *two's complement*, numbers are negated by complementing the bit pattern and *adding 1*, with overflow ignored. From 0 to $2^{N-1}-1$, positive numbers are assigned to binary values exactly as in one's complement. The remaining assignments (for the negative numbers) can be carried out using the two's complement negation rule. Regenerating the $N = 3$ example in this way gives Table 4.4.

Binary	Decimal
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Table 4.4: Three-bit two's-complement binary fixed-point numbers.

Note that according to our negation rule, $-(-4) = -4$. Logically, what has happened is that the result has "overflowed" and "wrapped around" back to itself. Note that $3 + 1 = -4$ also. In other words, if you compute 4 somehow, since there is no bit-pattern assigned to 4, you get -4, because -4 is assigned the bit pattern that would be assigned to 4 if N were larger. Note that numerical overflows naturally result in "wrap around" from positive to negative numbers (or from negative numbers to positive numbers). Computers normally "trap" overflows as an "exception." The exceptions are usually handled by a software "interrupt handler," and this can greatly slow down the processing by

the computer (one numerical calculation is being replaced by a rather sizable program).

Note that *temporary overflows* are ok in two's complement; that is, if you add 1 to 3 to get -4, adding -1 to -4 will give 3 again. This is why two's complement is a nice choice: it can be thought of as placing all the numbers on a "ring," allowing temporary overflows of intermediate results in a long string of additions and/or subtractions. All that matters is that the final sum lie within the supported dynamic range.

Computers designed with signal processing in mind (such as so-called "Digital Signal Processing (DSP) chips") generally just do the best they can without generating exceptions. For example, overflows quietly "saturate" instead of "wrapping around" (the hardware simply replaces the overflow result with the maximum positive or negative number, as appropriate, and goes on). Since the programmer may wish to know that an overflow has occurred, the first occurrence may set an "overflow indication" bit which can be manually cleared. The overflow bit in this case just says an overflow happened sometime since it was last checked.

Two's-Complement, Integer Fixed-Point Numbers

Let N denote the (even) number of bits. Then the value of a two's complement integer fixed-point number can be expressed in terms of its bits $\{b_i\}_{i=0}^{N-1}$ as

$$x = -b_0 \cdot 2^{N-1} + \sum_{i=1}^{N-2} b_i \cdot 2^{N-1-i}, \quad b_i \in \{0,1\} \tag{4.1}$$

We visualize the *binary word* containing these bits as

$$x = [b_0 b_1 \cdots b_{N-1}]$$

Each bit b_i is of course either 0 or 1. Check that the $N = 3$ table above is computed correctly using this formula. As an example, the numer 3 is expressed as

$$3 = [011] = -0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$$

while the number -3 is expressed as

$$-3 = [101] = -1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$$

and so on.

The most-significant bit in the word, b_0, can be interpreted as the "sign bit". If b_0 is "on", the number is negative. If it is "off", the number is either zero or positive.

The least-significant bit is b_{N-1}. "Turning on" that bit adds 1 to the number, and there are no fractions allowed.

The largest positive number is when all bits are on except b_0, in which case $x = 2^{N-1}-1$. The largest (in magnitude) negative number is $10\cdots0$, i.e., $b_0 = 1$ and $b_i = 0$ for all $i > 0$. Table 4.5 shows some of the most common cases.

N	$x_{\min}$	$x_{\max}$
8	-128	127
16	-32768	32767
24	-8,388,608	8,388,607
32	-2,147,483,648	2,147,483,647

Table 4.5: Numerical range limits in N-bit two's-complement.

4.3.3 Fractional Binary Fixed-Point Numbers

In "DSP chips" (microprocessors explicitly designed for digital signal processing applications), the most commonly used fixed-point format is *fractional fixed point*, also in two's complement.

Quite simply, fractional fixed-point numbers are obtained from integer fixed-point numbers by dividing them by 2^{N-1}. Thus, the only difference is a scaling of the assigned numbers. In the $N = 3$ case, we have the correspondences shown in Table 4.6.

4.3.4 How Many Bits are Enough for Digital Audio?

Armed with the above knowledge, we can visit the question "how many bits are enough" for digital audio. Since the threshold of hearing is around 0 db SPL, the "threshold of pain" is around 120 dB SPL, and each bit in a linear PCM format is worth about 6 dB of dynamic range, we find that we need $120/6 = 20$ bits to represent the full dynamic range of audio in a linear fixed-point format. This is a simplestic analysis because it is not quite right to equate the least-significant bit with the threshold of hearing; instead, we would like to adjust the *quantization noise floor* to just below

Binary	Decimal	
000	0	(0/4)
001	0.25	(1/4)
010	0.5	(2/4)
011	0.75	(3/4)
100	-1	(-4/4)
101	-0.75	(-3/4)
110	-0.5	(-2/4)
111	-0.25	(-1/4)

Table 4.6: Three-bit fractional fixed-point numbers.

the threshold of hearing. Since the threshold of hearing is non-uniform, we would also prefer a *shaped* quantization noise floor (a feat that can be accomplished using *filtered error feedback*[10] Nevertheless, the simplistic result gives an answer similar to the more careful analysis, and 20 bits is a good number. However, this still does not provide for *headroom* needed in a digital recording scenario. We also need both headroom and *guard bits* on the lower end when we plan to carry out a lot of signal processing operations, especially digital filtering. As an example, a 1024-point FFT (Fast Fourier Transform) can give amplitudes 1024 times the input amplitude (such as in the case of a constant "dc" input signal), thus requiring 10 headroom bits. In general, 24 fixed-point bits are pretty reasonable to work with, although you still have to scale very carefully, and 32 bits are preferable.

4.3.5 When Do We Have to Swap Bytes?

When moving a soundfile from one computer to another, such as from a "PC" to a "Mac" (Intel processor to Motorola processor), the bytes in each sound sample have to be *swapped*. This is because Motorola processors are *big endian* (bytes are numbered from most-significant to least-significant in a multi-byte word) while Intel processors are *little en-*

[10]Normally, quantization error is computed as $e(n) = x(n) - \hat{x}(n)$, where $x(n)$ is the signal being quantized, and $\hat{x}(n) = Q[x(n)]$ is the quantized value, obtained by rounding to the nearest representable amplitude. Filtered error feedback uses instead the formula $\hat{x}(n) = Q[x(n) + \mathcal{L}\{e(n-1)\}]$, where $\mathcal{L}\{\ \}$ denotes a filtering operation which "shapes" the quantization noise spectrum. An excellent article on the use of round-off error feedback in audio digital filters is [9].

dian (bytes are numbered from least-significant to most-significant).[11] Any Mac program that supports a soundfile format native to PCs (such as `.wav` files) will swap the bytes for you. You only have to worry about swapping the bytes yourself when reading `raw binary` soundfiles from a foreign computer, or when digging the sound samples out an "unsupported" soundfile format yourself.

Since soundfiles typically contain 16 bit samples (not for any good reason, as we now know), there are only two bytes in each audio sample. Let L denote the least-significant byte, and M the most-significant byte. Then a 16-bit word is most naturally written $[M, L] = M \cdot 256 + L$, i.e., the most-significant byte is most naturally written to the left of the least-significant byte, analogous to the way we write binary or decimal integers. This "most natural" ordering is used as the byte-address ordering in big-endian processors:

M,L, M,L, M,L, ..., M,L (Big Endian)

Little-endian machines, on the other hand, store bytes in the order

L,M, L,M, L,M, ..., L,M. (Little Endian)

These orderings are preserved when the sound data are written to a disk file.

Since a byte (eight bits) is the smallest addressable unit in modern day processor families, we don't have to additionally worry about reversing the *bits* in each byte. Bits are not given explicit "addresses" in memory. They are extracted by means other than simple addressing (such as masking and shifting operations, table look-up, or using specialized processor instructions).

Table 4.7 lists popular present-day processors and their "endianness":[12] When compiling C or C++ programs under UNIX, there may be a BYTE_ORDER macro in endian.h or bytesex.h. In other cases, there may be a defined macro __INTEL__, __LITTLE_ENDIAN__, __BIG_ENDIAN__, or the like.

[11] Remember that byte addresses in a big endian word start at the big end of the word, while in a little endian architecture, they start at the little end of the word.

[12] Thanks to Bill Schottstaedt for help with this table.

Processor Family	Endian
Pentium (Intel)	Little
Alpha (DEC/Compaq)	Little
680x0 (Motorola)	Big
PowerPC (Motorola & IBM)	Big
SPARC (Sun)	Big
MIPS (SGI)	Big

Table 4.7: Byte ordering in the major computing platforms.

4.4 Logarithmic Number Systems for Audio

Since hearing is approximately logarithmic, it makes sense to represent sound samples in a logarithmic or semi-logarithmic number format. Floating-point numbers in a computer are partially logarithmic (the exponent part), and one can even use an entirely logarithmic fixed-point number system. The μ-law amplitude-encoding format is linear at small amplitudes and becomes logarithmic at large amplitudes. This section discusses these formats.

4.4.1 Floating-Point Numbers

Floating-point numbers consist of an "exponent," "significand", and "sign bit". For a negative number, we may set the sign bit of the floating-point word and negate the number to be encoded, leaving only nonnegative numbers to be considered. Zero is represented by all zeros, so now we need only consider positive numbers.

The basic idea of floating point *encoding* of a binary number is to *normalize* the number by *shifting* the bits either left or right until the shifted result lies between 1/2 and 1. (A left-shift by one place in a binary word corresponds to multiplying by 2, while a right-shift one place corresponds to dividing by 2.) The number of bit-positions shifted to normalize the number can be recorded as a signed integer. The negative of this integer (i.e., the shift required to recover the original number) is defined as the *exponent* of the floating-point encoding. The normalized number between 1/2 and 1 is called the *significand*, so called because it holds all the "significant bits" of the number.

Floating point notation is exactly analogous to "scientific notation" for decimal numbers, e.g., 1.2345×10^{-9}; the number of significant digits,

5 in this example, is determined by counting digits in the "significand" 1.2345, while the "order of magnitude" is determined by the power of 10 (-9 in this case). In floating-point numbers, the significand is stored in fractional two's-complement binary format, and the exponent is stored as a binary integer.

Since the significand lies in the interval $[1/2, 1)$,[13] its most significant bit is always a 1, so it is not actually stored in the computer word, giving one more significant bit of precision.

Let's now restate the above a little more precisely. Let $x > 0$ denote a number to be encoded in floating-point, and let $\tilde{x} = x \cdot 2^{-E}$ denote the normalized value obtained by shifting x either E bits to the right (if $E > 0$), or $|E|$ bits to the left (if $E < 0$). Then we have $1/2 \leq \tilde{x} < 1$, and $x = \tilde{x} \cdot 2^E$. The *significand* M of the floating-point representation for x is defined as the binary encoding of $\tilde{x}$.[14] It is often the case that $\tilde{x}$ requires more bits than are available for exact encoding. Therefore, the significand is typically *rounded* (or truncated) to the value closest to $\tilde{x}$. Given N_M bits for the significand, the encoding of $\tilde{x}$ can be computed by multiplying it by 2^{N_M} (left-shifting it N_M bits), *rounding* to the nearest integer (or *truncating* toward minus infinity—the as implemented by the `floor()` function), and encoding the N_M-bit result as a binary (signed) integer.

As a final practical note, exponents in floating-point formats may have a *bias*. That is, instead of storing E as a binary integer, you may find a binary encoding of $E - B$ where B is the bias.[15]

These days, floating-point formats generally follow the IEEE standards set out for them. A single-precision floating point word is 32 bits (four bytes) long, consisting of 1 sign bit, 8 exponent bits, and 23 significand bits, normally laid out as

S EEEEEEEE MMMMMMMMMMMMMMMMMMMMMMM

[13] The notation $[a, b)$ denotes a *half-open interval* which includes a but not b.

[14] Another term commonly heard for "significand" is "mantissa." However, this use of the term "mantissa" is not the same as its previous definition as the fractional part of a logarithm. We will therefore use only the term "significand" to avoid confusion.

[15] By choosing the bias equal to half the numerical dynamic range of E (thus effectively inverting the sign bit of the exponent), it becomes easier to compare two floating-point numbers in hardware: the entire floating-point word can be treated by the hardware as one giant integer for numerical comparison purposes. This works because negative exponents correspond to floating-point numbers less than 1 in magnitude, while. positive exponents correspond to floating-point numbers greater than 1 in magnitude.

where S denotes the sign bit, E an exponent bit, and M a significand bit. Note that in this layout, ordinary integer comparison can be used in the hardware.

A double-precision floating point word is 64 bits (eight bytes) long, consisting of 1 sign bit, 11 exponent bits, and 52 significand bits. In the Intel Pentium processor, there is also an *extended precision* format, used for intermediate results, which is 80 bits (ten bytes) containing 1 sign bit, 15 exponent bits, and 64 significand bits. In Intel processors, the exponent bias is 127 for single-precision floating-point, 1023 for double-precision, and 16383 for extended-precision. The single and double precision formats have a "hidden" significand bit, while the extended precision format does not. Thus, the most significant significand bit is always set in extended precision.

The MPEG-4 audio compression standard (which supports compression using music synthesis algorithms) specifies that the numerical calculations in any MPEG-4 audio decoder should be at least as accurate as 32-bit single-precision floating point.

4.4.2 Logarithmic Fixed-Point Numbers

In some situations it makes sense to use *logarithmic fixed-point*. This number format can be regarded as a floating-point format consisting of an exponent and no explicit significand. However, the exponent is not interpreted as an integer as it is in floating point. Instead, it has a fractional part which is a true *mantissa*. (The integer part is then the "characteristic" of the logarithm.) In other words, a logarithmic fixed-point number is a binary encoding of the log-base-2 of the signal-sample magnitude. The sign bit is of course separate.

An example 16-bit logarithmic fixed-point number format suitable for digital audio consists of one sign bit, a 5-bit characteristic, and a 10-bit mantissa:

S CCCCC MMMMMMMMMM

The 5-bit characteristic gives a dynamic range of about 6 dB $\times 2^5 = 192$ dB. This is an excellent dynamic range for digital audio. (While 120 dB would seem to be enough for audio, consider that when digitally modeling a brass musical instrument, say, the internal air pressure near the "virtual mouthpiece" can be far higher than what actually reaches the ears in the audience.)

A nice property of logarithmic fixed-point numbers is that multiplies simply become additions and divisions become subtractions. The hard elementary operation are now addition and subtraction, and these are normally done using table lookups to keep them simple.

One "catch" when working with logarithmic fixed-point numbers is that you can't let "dc" build up. A wandering dc component will cause the quantization to be coarse even for low-level "ac" signals. It's a good idea to make sure dc is always filtered out in logarithmic fixed-point.

4.4.3 Mu-Law Companding

A *companding* operation *compresses* dynamic range on encode and *expands* dynamic range on decode. In digital telephone networks and voice modems (currently in use everywhere), standard *CODEC*[16] chips are used in which audio is digitized in a simple 8-bit *μ-law format* (or simply "mu-law").

Given an input sample $x(n)$ represented in some internal format, such as a `short`, it is converted to 8-bit mu-law format by the formula [10]

$$\hat{x}_\mu \stackrel{\Delta}{=} Q_\mu\left[\log_2\left(1+\mu\left|x(n)\right|\right)\right]$$

where $Q_\mu[]$ is a *quantizer* which produces a kind of logarithmic fixed-point number with a 3-bit characteristic and a 4-bit mantissa, using a small table lookup for the mantissa.

As we all know from talking on the telephone, mu-law sounds really quite good for voice, at least as far as *intelligibility* is concerned. However, because the telephone bandwidth is only around 3 kHz (nominally 200–3200 Hz), there is very little "bass" and no "highs" in the spectrum above 4 kHz. This works out fine for intelligibility of voice because the first three *formants* (envelope peaks) in typical speech spectra occur in this range, and also because the difference in spectral shape (particularly at high frequencies) between consonants such as "sss", "shshsh", "fff", "thththh", etc., are sufficiently preserved in this range. As a result of the narrow bandwidth provided for speech, it is sampled at only 8 kHz in standard CODEC chips.

For "wideband audio", we like to see sampling rates at least as high as 44.1 kHz, and the latest systems are moving to 96 kHz (mainly because oversampling simplifies signal processing requirements in various areas,

[16] CODEC is an acronym for "COder/DECoder".

not because we can actually hear anything above 20 kHz). In addition, we like the low end to extend at least down to 20 Hz or so. (The lowest note on a normally tuned bass guitar is E1 = 41.2 Hz. The lowest note on a grand piano is A0 = 27.5 Hz.)

4.5 Appendix A: Round-Off Error Variance

This section shows how to derive that the noise power of quantization error is $q^2/12$, where q is the quantization step size.

Each round-off error in quantization noise $e(n)$ is modeled as a uniform random variable between $-q/2$ and $q/2$. It therefore has the *probability density function* (pdf)

$$p_e(x) = \begin{cases} \frac{1}{q}, & |x| \leq \frac{q}{2} \\ 0, & |x| > \frac{q}{2} \end{cases}$$

Thus, the *probability* that a given roundoff error $e(n)$ lies in the interval $[x_1, x_2]$ is given by

$$\int_{x_1}^{x_2} p_e(x)dx = \frac{x_2 - x_1}{q}$$

assuming of course that x_1 and x_2 lie in the allowed range $[-q/2, q/2]$. We might loosely refer to $p_e(x)$ as a *probability distribution*, but technically it is a probability *density* function, and to obtain probabilities, we have to integrate over one or more intervals, as above. We use probability *distributions* for variables which take on *discrete* values (such as dice), and we use probability *densities* for variables which take on *continuous* values (such as round-off errors).

The *mean* of a random variable is defined as

$$\mu_e \stackrel{\Delta}{=} \int_{-\infty}^{\infty} x p_e(x)dx = 0$$

In our case, the mean is zero because we are assuming the use of *rounding* (as opposed to truncation, etc.).

The *mean of a signal* $e(n)$ is the same thing as the *expected value* of $e(n)$, which we write as $\mathcal{E}\{e(n)\}$. In general, the expected value of *any* function $f(v)$ of a random variable v is given by

$$\mathcal{E}\{f(v)\} \stackrel{\Delta}{=} \int_{-\infty}^{\infty} f(x) p_v(x)dx$$

Since the quantization-noise signal $e(n)$ is modeled as a series of independent, identically distributed (iid) random variables, we can *estimate* the mean by *averaging* the signal over time. Such an estimate is called a *sample mean*.

Probability distributions are often be characterized by their *moments*. The nth moment of the pdf $p(x)$ is defined as

$$\int_{-\infty}^{\infty} x^n p(x) dx$$

Thus, the mean $\mu_x = \mathcal{E}\{e(n)\}$ is the *first* moment of the pdf. The second moment is simply the expected value of the random variable squared, i.e., $\mathcal{E}\{e^2(n)\}$.

The *variance* of a random variable $e(n)$ is defined as the *second central moment* of the pdf:

$$\sigma_e^2 \stackrel{\Delta}{=} \mathcal{E}\{[e(n) - \mu_e]^2\} = \int_{-\infty}^{\infty} (x - \mu_e)^2 p_e(x) dx$$

"Central" just means that the moment is evaluated after subtracting out the *mean*, that is, looking at $e(n) - \mu_e$ instead of $e(n)$. In the case of round-off errors, the mean is zero, so subtracting out the mean has no effect. Plugging in the constant pdf for our random variable $e(n)$ which we assume is uniformly distributed on $[-q/2, q/2]$, we obtain the variance

$$\sigma_e^2 = \int_{-q/2}^{q/2} x^2 \frac{1}{q} dx = \frac{1}{q} \frac{1}{3} x^3 \Big|_{-q/2}^{q/2} = \frac{q^2}{12}$$

Note that the variance of $e(n)$ can be estimated by averaging $e^2(n)$ over time, that is, by computing the *mean square*. Such an estimate is called the *sample variance*. For sampled physical processes, the sample variance is proportional to the *average power* in the signal. Finally, the square root of the sample variance (the *rms level*) is sometimes called the *standard deviation* of the signal, but this term is only precise when the random variable has a Gaussian pdf.

Some good textbooks in the area of statistical signal processing include [11, 12, 13]. EE 278 is the starting course on this topic at Stanford.

4.6 Appendix B: Electrical Engineering 101

The state of an ideal resistor is completely specified by the voltage across it (call it V *volts*) and the current passing through it (I Amperes, or

simply "amps"). The ratio of voltage to current gives the value of the resistor ($V/I = R$ = resistance in Ohms). The fundamental relation between voltage and current in a resistor is called *Ohm's Law*:

$$V(t) = R \cdot I(t) \qquad \text{(Ohm's Law)}$$

where we have indicated also that the voltage and current may vary with time (while the resistor value normally does not).

The electrical *power* in *watts* dissipated by a resistor R is given by

$$\mathcal{P} = V \cdot I = \frac{V^2}{R} = R \cdot I^2$$

where V is the voltage and I is the current. Thus, volts times amps gives watts. Also, volts squared over ohms equals watts, and so on.

Chapter 5

Sinusoids and Exponentials

This chapter provides an introduction to sinusoids, exponentials, complex sinusoids, t_{60}, in-phase and quadrature sinusoidal components, the analytic signal, positive and negative frequencies, constructive and destructive interference, invariance of sinusoidal frequency in linear time-invariant systems, circular motion as the vector sum of in-phase and quadrature sinusoidal motions, sampled sinusoids, generating sampled sinusoids from powers of z, and plot examples using Mathematica.

5.1 Sinusoids

A *sinusoid* is any function of time having the following form:

$$x(t) = A\sin(\omega t + \phi)$$

where all variables are real numbers, and

$$\begin{aligned} A &= \text{Peak Amplitude (nonnegative)} \\ \omega &= \text{Radian Frequency (rad/sec)} \\ &= 2\pi f \ (f \text{ in Hz}) \\ t &= \text{Time (sec)} \\ f &= \text{Frequency (Hz)} \\ \phi &= \text{Phase (radians)} \end{aligned}$$

The term "peak amplitude" is often shortened to "amplitude," e.g., "the amplitude of the sound was measured to be 5 Pascals." Strictly speaking, however, the "amplitude" of a signal x is its instantaneous

value $x(t)$ at any time t. The peak amplitude A satisfies $x(t) \leq A$. The "instantaneous magnitude" or simply "magnitude" of a signal $x(t)$ is given by $|x(t)|$, and the peak magnitude is the same thing as the peak amplitude.

Note that Hz is an abbreviation for Hertz which physically means "cycles per second." You might also encounter the older (and clearer) notation "c.p.s." for cycles per second.

Since $\sin(\theta)$ is periodic with period 2π, the phase $\phi \pm 2\pi$ is indistinguishable from the phase ϕ. As a result, we may restrict the range of ϕ to any length 2π interval. When needed, we will choose

$$-\pi \leq \phi < \pi,$$

i.e., $\phi \in [-\pi, \pi)$. You may also encounter the convention $\phi \in [0, 2\pi)$.

5.1.1 Example Sinusoids

Figure 5.1 plots the sinusoid $A\sin(2\pi ft+\phi)$, for $A = 10$, $f = 2.5$, $\phi = \pi/4$, and $t \in [0, 1]$. Study the plot to make sure you understand the effect of changing each parameter (amplitude, frequency, phase), and also note the definitions of "peak-to-peak amplitude" and "zero crossings."

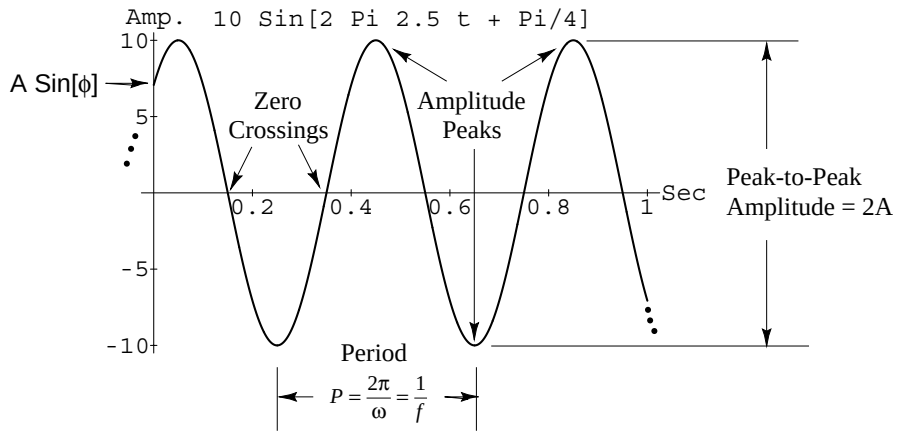

Figure 5.1: An example sinusoid.

The Mathematica code for generating this figure is listed in §5.4.

A "tuning fork" vibrates approximately sinusoidally. An "A-440" tuning fork oscillates at 440 cycles per second. As a result, a tone recorded

from an ideal A-440 tuning fork is a sinusoid at $f = 440$ Hz. The amplitude A determines how loud it is and depends on how hard we strike the tuning fork. The phase ϕ is set by exactly *when* we strike the tuning fork (and on our choice of when time 0 is). If we record an A-440 tuning fork on an analog tape recorder, the electrical signal recorded on tape is of the form

$$x(t) = A\sin(2\pi 440t + \phi)$$

As another example, the sinusoid at amplitude 1 and phase $\pi/2$ (90 degrees) is simply

$$x(t) = \sin(\omega t + \pi/2) = \cos(\omega t)$$

Thus, $\cos(\omega t)$ is a sinusoid at phase 90-degrees, while $\sin(\omega t)$ is a sinusoid at zero phase. Note, however, that we could just as well have defined $\cos(\omega t)$ to be the zero-phase sinusoid rather than $\sin(\omega t)$. It really doesn't matter, except to be consistent in any given usage. The concept of a "sinusoidal signal" is simply that it is equal to a sine or cosine function at some amplitude, frequency, and phase. It does not matter whether we choose sin() or cos() in the "official" definition of a sinusoid. You may encounter both definitions. Using sin() is nice since "sinusoid" in a sense generalizes sin(). However, using cos() is nicer when defining a sinusoid to be the real part of a complex sinusoid (which we'll talk about later).

5.1.2 Why Sinusoids are Important

Sinusoids are fundamental in a variety of ways.

One reason for the importance of sinusoids is that they are *fundamental in physics*. Anything that resonates or oscillates produces quasi-sinusoidal motion. See *simple harmonic motion* in any freshman physics text for an introduction to this topic.

Another reason sinusoids are important is that they are *eigenfunctions of linear systems* (which we'll say more about later). This means that they are important for the analysis of *filters* such as reverberators, equalizers, certain (but not all) "effects", etc.

Perhaps most importantly, from the point of view of computer music research, is that the human *ear* is a kind of *spectrum analyzer*. That is, the chochlea of the inner ear physically splits sound into its (near) sinusoidal components. This is accomplished by the *basilar membrane* in the inner ear: a sound wave injected at the *oval window* (which is connected

via the bones of the middle ear to the *ear drum*), travels along the basilar membrane inside the coiled cochlea. The membrane starts out thick and stiff, and gradually becomes thinner and more compliant toward its apex (the *helicotrema*). A stiff membrane has a high resonance frequency while a thin, compliant membrane has a low resonance frequency (assuming comparable mass density, or at least less of a difference in mass than in compliance). Thus, as the sound wave travels, each frequency in the sound resonates at a particular *place* along the basilar membrane. The highest frequencies resonate right at the entrance, while the lowest frequencies travel the farthest and resonate near the helicotrema. The membrane resonance effectively "shorts out" the signal energy at that frequency, and it travels no further. Along the basilar membrane there are *hair cells* which "feel" the resonant vibration and transmit an increased firing rate along the auditory nerve to the brain. Thus, the ear is very literally a Fourier analyzer for sound, albeit nonlinear and using "analysis" parameters that are difficult to match exactly. Nevertheless, by looking at spectra (which display the amount of each sinusoidal frequency present in a sound), we are looking at a representation much more like what the brain receives when we hear.

5.1.3 In-Phase and Quadrature Sinusoidal Components

From the trig identity $\sin(A+B) = \sin(A)\cos(B) + \cos(A)\sin(B)$, we have

$$\begin{aligned} x(t) &= A\sin(\omega t + \phi) = A\sin(\phi + \omega t) \\ &= [A\sin(\phi)]\cos(\omega t) + [A\cos(\phi)]\sin(\omega t) \\ &\stackrel{\Delta}{=} A_1\cos(\omega t) + A_2\sin(\omega t) \end{aligned}$$

From this we may conclude that every sinusoid can be expressed as the sum of a sine function (phase zero) and a cosine function (phase $\pi/2$). If the sine part is called the "in-phase" component, the cosine part can be called the "phase-quadrature" component. In general, "phase quadrature" means "90 degrees out of phase," i.e., a relative phase shift of $\pm\pi/2$.

It is also the case that every sum of an in-phase and quadrature component can be expressed as a single sinusoid at some amplitude and phase. The proof is obtained by working the previous derivation backwards.

Figure 5.2 illustrates in-phase and quadrature components overlaid. Note that they only differ by a relative 90 degree phase shift. (See §5.4 for the Mathematica code for this figure.)

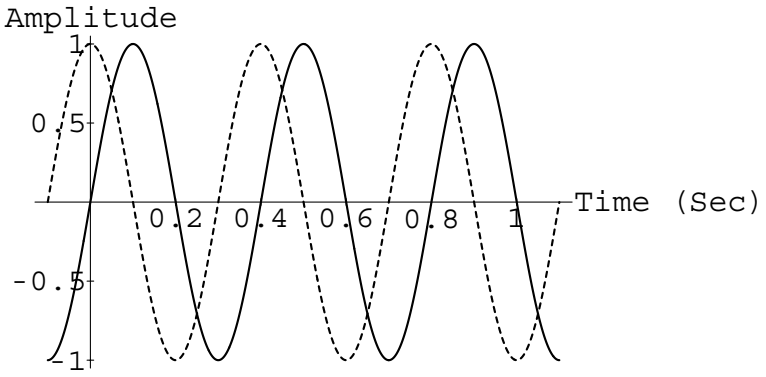

Figure 5.2: In-phase and quadrature sinusoidal components.

5.1.4 Sinusoids at the Same Frequency

An important property of sinusoids at a particular frequency is that they are *closed* with respect to addition. In other words, if you take a sinsusoid, make many copies of it, scale them all by different gains, delay them all by different amounts, and add them up, you always get a sinusoid at the same original frequency. This is a nontrivial property. It obviously holds for any constant signal $x(t) = c$ (which we may regard as a sinusoid at frequency $f = 0$), but it is not obvious for $f \neq 0$ (see Fig. 5.2 and think about the sum of the two waveforms shown being precisely a sinusoid).

Since every linear, time-invariant (LTI[1]) system (filter) operates by copying, scaling, delaying, and summing its input signal(s) to create its output signal(s), it follows that when a sinusoid at a particular frequency is input to an LTI system, a sinusoid at that same frequency always appears at the output. Only the amplitude and phase can be changed by the system. We say that sinusoids are *eigenfunctions* of LTI systems. Conversely, if the system is nonlinear or time-varying, new frequencies are created at the system output.

To prove the important invariance property of sinusoids, we may simply express all scaled and delayed sinusoids in the "mix" in terms of their in-phase and quadrature components and then add them up. For example, consider the case of two sinusoids arbitrarily scaled by gains $g1, g2$

[1]A system S is said to be *linear* if for any two input signals $x_1(t)$ and $x_2(t)$, we have $S[x_1(t) + x_2(t)] = S[x_1(t)] + S[x_2(t)]$. A system is said to be *time invariant* if $S[x(t-\tau)] = y(t-\tau)$, where $y(t) \triangleq S[x(t)]$. This subject is developed in detail in [4].

and arbitrarily delayed by time-delays $t1, t2$:

$$\begin{aligned} y(t) &\stackrel{\Delta}{=} g_1 x(t-t_1) + g_2 x(t-t_2) \\ &= g_1 A \sin[\omega(t-t_1)+\phi] + g_2 A \sin[\omega(t-t_2)+\phi] \end{aligned}$$

Focusing on the first term, we have

$$\begin{aligned} g_1 A \sin[\omega(t-t_1)+\phi] &= g_1 A \sin[\omega t + (\phi - \omega t_1)] \\ &= [g_1 A \sin(\phi - \omega t_1)]\cos(\omega t) + [g_1 A \cos(\phi - \omega t_1)]\sin(\omega t) \\ &\stackrel{\Delta}{=} A_1 \cos(\omega t) + B_1 \sin(\omega t) \end{aligned}$$

We similarly compute

$$g_2 A \sin[\omega(t-t_2)+\phi] = A_2 \cos(\omega t) + B_2 \sin(\omega t)$$

and add to obtain

$$y(t) = (A_1 + A_2)\cos(\omega t) + (B_1 + B_2)\sin(\omega t)$$

This result, consisting of one in-phase and one quadrature signal component, can now be converted to a single sinusoid at some amplitude and phase (and frequency ω), as discussed above.

5.1.5 Constructive and Destructive Interference

Sinusoidal signals are analogous to monochromatic laser light. You might have seen "speckle" associated with laser light, caused by destructive interference of multiple reflections of the light beam. In a room, the same thing happens with sinusoidal sound. For example, play a simple sinusoidal tone (e.g., "A-440" which is a sinusoid at frequency $f = 440$ Hz) and walk around the room with one ear plugged. If the room is reverberant you should be able find places where the sound goes completely away due to destructive interference. In between such places (which we call "nodes" in the soundfield), there are "antinodes" at which the sound is louder by 6 dB (amplitude doubled) due to constructive interference. In a diffuse reverberant soundfield, the distance between nodes is on the order of a wavelength (the "correlation distance" within the random soundfield).

The way reverberation produces nodes and antinodes for sinusoids in a room is illustrated by the simple *comb filter*.[2] There is also shown in Fig. 5.3. A unit-amplitude sinusoid is present at the input, and the output must also be sinusoidal, since the comb filter is linear and time-invariant. The feedforward path of the comb filter has a gain of 0.5, and the delay is one period in one case and half a period in the other. With the delay set to one period, the unit amplitude sinusoid coming out of the delay line *constructively interferes* with the amplitude 0.5 sinusoid from the feed-forward path, and the output amplitude is therefore $1 + 0.5 = 1.5$. In the other case, with the delay set to *half* period, the unit amplitude sinusoid coming out of the delay line *destructively interferes* with the amplitude 0.5 sinusoid from the feed-forward path, and the output amplitude therefore drops to $|-1 + 0.5| = 0.5$.

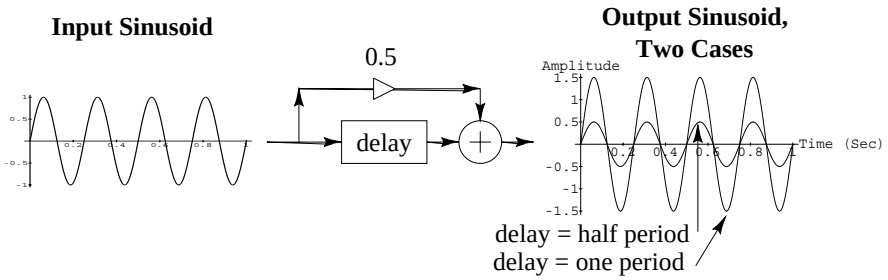

Figure 5.3: A comb filter with a sinusoidal input.

Consider a fixed delay of τ seconds for the delay line. Constructive interference happens at all frequencies for which an *exact integer* number of periods fits in the delay line, i.e., $f\tau = 0, 1, 2, 3, \ldots$, or $f = n/\tau$, for $n = 0, 1, 2, 3, \ldots$. On the other hand, destructive interference happens

[2]Technically, this is the *feedforward comb filter*, also called the "inverse comb filter" [14]. The longer names are meant to distinguish it from the *feedback comb filter* (defined as "the" comb filter in Dodge and Jerse [15]). In the feedback comb filter, the delay output is fed *back* around the delay line and summed with the delay input instead of the input being fed *forward* around the delay line and summed with its output. The frequency response of the feedforward comb filter is the inverse of that of the feedback comb filter (one will cancel the effect of the other), hence the name "inverse comb filter." When the delay in the feedforward comb filter is varied slowly over time, the *flanger effect* is obtained. Flanging was originally achieved by mixing the outputs of two LP record turntables and changing their relative speeds by alternately touching the "flange" of each turntable to slow it down.

at all frequencies for which number of periods in the delay line is an integer plus a half, i.e., $f\tau = 1.5, 2.5, 3.5$, etc., or, $f = (n + 1/2)/\tau$, for $n = 0, 1, 2, 3, \ldots$. It is quick to verify that frequencies of constructive interference alternate with frequencies of destructive interference, and therefore the *amplitude response* of the comb filter (a plot of gain versus frequency) looks as shown in Fig. 5.4.

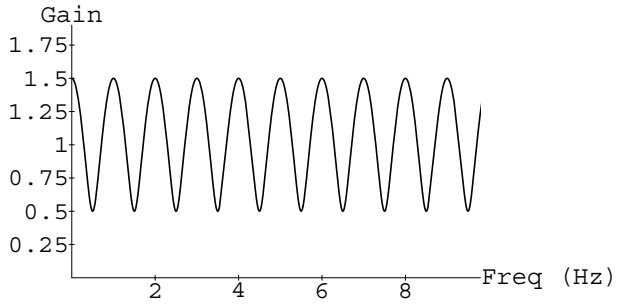

Figure 5.4: Comb filter amplitude response when delay $\tau = 1$ sec.

The ampitude response of a comb filter has a "comb" like shape, hence the name.[3] Note that if the feedforward gain is increased from 0.5 to 1, the comb-filter gain ranges between 0 (complete cancellation) and 2. Negating the feedforward gain inverts the gain curve, placing a minumum at dc[4] instead of a peak.

5.2 Exponentials

The canonical form of an exponential function, as typically used in signal processing, is

$$a(t) = Ae^{-t/\tau}, \quad t \geq 0$$

where τ is called the *time constant* of the exponential. A is the peak amplitude, as before. The time constant is the time it takes to decay by

[3] While there is no reason it should be obvious at this point, the comb-filter gain varies in fact sinusoidally between 0.5 and 1.5. It looks more "comb" like on a dB amplitude scale, which is more appropriate for audio applications.

[4] "dc" means "direct current" and is an electrical engineering term for "frequency 0".

$1/e$, i.e.,

$$\frac{a(\tau)}{a(0)} = \frac{1}{e}$$

A normalized exponential decay is depicted in Fig. 5.5.

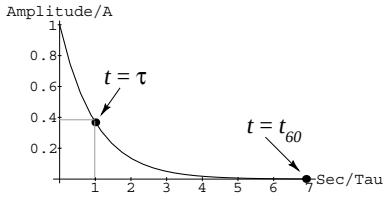

Figure 5.5: The decaying exponential $Ae^{-t/\tau}$.

5.2.1 Why Exponentials are Important

Exponential *decay* occurs naturally when a quantity is decaying at a rate which is proportional to how much is left. In nature, all *linear resonators*, such as musical instrument strings and woodwind bores, exhibit exponential decay in their response to a momentary excitation. As another example, reverberant energy in a room decays exponentially after the direct sound stops. Essentially all *undriven oscillations* decay exponentially (provided they are linear and time-invariant). Undriven means there is no ongoing source of driving energy. Examples of undriven oscillations include the vibrations of a tuning fork, struck or plucked strings, a marimba or xylophone bar, and so on. Examples of driven oscillations include horns, woodwinds, bowed strings, and voice. Driven oscillations must be periodic while undriven oscillations normally are not, except in idealized cases.

Exponential *growth* occurs when a quantity is increasing at a rate proportional to the current amount. Exponential growth is *unstable* since nothing can grow exponentially forever without running into some kind of limit. Note that a positive time constant corresponds to exponential decay, while a negative time constant corresponds to exponential growth. In signal processing, we almost always deal exclusively with exponential decay (positive time constants).

Exponential growth and decay are illustrated in Fig. 5.6.

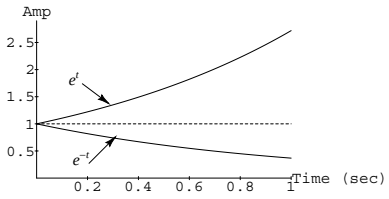

Figure 5.6: Growing and decaying exponentials.

5.2.2 Audio Decay Time (T60)

In audio, a decay by $1/e$ is too small to be considered a practical "decay time." In *architectural acoustics* (which includes the design of concert halls), a more commonly used measure of decay is "t_{60}" (or `T60`), which is defined as the *time to decay by* 60 *dB*.[5] That is, t_{60} is obtained by solving the equation

$$\frac{a(t_{60})}{a(0)} = 10^{-60/20} = 0.001$$

Using the definition of the exponential $a(t) = Ae^{-t/\tau}$, we find

$$t_{60} = \ln(1000)\tau \approx 6.91\tau$$

Thus, t_{60} is about seven time constants. See where t_{60} is marked on Fig. 5.5 compared with τ.

5.3 Complex Sinusoids

Recall Euler's Identity,

$$e^{j\theta} = \cos(\theta) + j\sin(\theta)$$

Multiplying this equation by $A \geq 0$ and setting $\theta = \omega t + \phi$, we obtain the definition of the *complex sinusoid*:

$$s(t) \stackrel{\Delta}{=} Ae^{j(\omega t+\phi)} = A\cos(\omega t + \phi) + jA\sin(\omega t + \phi)$$

[5]Recall that a gain factor g is converted to *decibels* (dB) by the formula $20\log_{10}(g)$.

Thus, a complex sinusoid consists of an in-phase component for its real part, and a phase-quadrature component for its imaginary part. Since $\sin^2(\theta) + \cos^2(\theta) = 1$, we have

$$|s(t)| \equiv A$$

That is, the complex sinusoid is *constant modulus*. (The symbol "$\equiv$" means "identically equal to," i.e., for all t.) The phase of the complex sinusoid is

$$\angle s(t) = \omega t + \phi$$

The derivative of the phase of the complex sinusoid gives its frequency

$$\frac{d}{dt}\angle s(t) = \omega = 2\pi f$$

5.3.1 Circular Motion

Since the modulus of the complex sinusoid is constant, it must lie on a *circle* in the complex plane. For example,

$$x(t) = e^{j\omega t}$$

traces out *counter-clockwise* circular motion along the unit circle in the complex plane, while

$$\overline{x(t)} = e^{-j\omega t}$$

is *clockwise* circular motion.

We call a complex sinusoid of the form $e^{j\omega t}$, where $\omega > 0$, a *positive-frequency sinusoid*. Similarly, we define a complex sinusoid of the form $e^{-j\omega t}$, with $\omega > 0$, to be a *negative-frequency sinusoid*. Note that a positive- or negative-frequency sinusoid is necessarily complex.

5.3.2 Projection of Circular Motion

We have

$$\begin{aligned} \mathrm{re}\left\{e^{j\omega t}\right\} &= \cos(\omega t) \\ \mathrm{im}\left\{e^{j\omega t}\right\} &= \sin(\omega t) \end{aligned}$$

Interpreting this in the complex plane tells us that *sinusoidal motion is the projection of circular motion onto any straight line.* Thus, the sinusoidal motion $\cos(\omega t)$ is the projection of the circular motion $e^{j\omega t}$

onto the x (real-part) axis, while $\sin(\omega t)$ is the projection of $e^{j\omega t}$ onto the y (imaginary-part) axis.

Figure 5.7 shows a plot of a complex sinusoid versus time, along with its projections onto coordinate planes. This is a 3D plot showing the z-plane versus time. The axes are the real part, imaginary part, and time. (Or we could have used magnitude and phase versus time.)

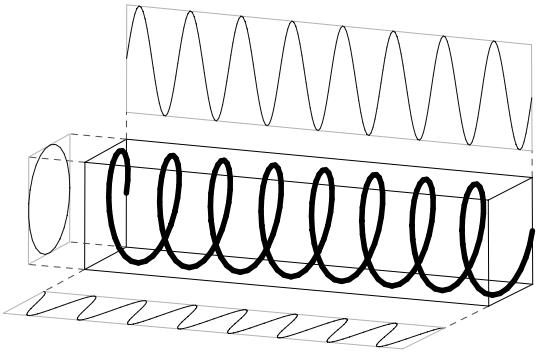

Figure 5.7: A complex sinusoid and its projections.

Note that the left projection (onto the z plane) is a circle, the lower projection (real-part vs. time) is a cosine, and the upper projection (imaginary-part vs. time) is a sine. A point traversing the plot projects to uniform circular motion in the z plane, and sinusoidal motion on the two other planes.

5.3.3 Positive and Negative Frequencies

Earlier, we used Euler's Identity to show

$$\begin{aligned}\cos(\theta) &= \frac{e^{j\theta} + e^{-j\theta}}{2} \\ \sin(\theta) &= \frac{e^{j\theta} - e^{-j\theta}}{2j}\end{aligned}$$

Setting $\theta = \omega t + \phi$, we see that both sine and cosine (and hence all real sinusoids) consist of a sum of equal and opposite circular motion. Phrased differently, every real sinusoid consists of an equal contribution of positive and negative frequency components. This is true of all real signals. When we get to spectrum analysis, we will find that every real signal contains equal amounts of positive and negative frequencies, i.e.,

if $X(\omega)$ denotes the spectrum of the real signal $x(t)$, we will always have $|X(-\omega)| = |X(\omega)|$.

Note that, mathematically, the complex sinusoid $Ae^{j(\omega t+\phi)}$ is really *simpler* and *more basic* than the real sinusoid $A\sin(\omega t + \phi)$ because $e^{j\omega t}$ consists of one frequency ω while $\sin(\omega t)$ really consists of two frequencies ω and $-\omega$. We may think of a real sinusoid as being the sum of a positive-frequency and a negative-frequency complex sinusoid, so in that sense real sinusoids are "twice as complicated" as complex sinusoids. Complex sinusoids are also nicer because they have a *constant modulus*. "Amplitude envelope detectors" for complex sinusoids are trivial: just compute the square root of the sum of the squares of the real and imaginary parts to obtain the *instantaneous peak amplitude* at any time. Frequency demodulators are similarly trivial: just differentiate the phase of the complex sinusoid to obtain its *instantaneous frequency*. It should therefore come as no surprise that signal processing engineers often prefer to convert real sinusoids into complex sinusoids before processing them further.

5.3.4 The Analytic Signal and Hilbert Transform Filters

A signal which has no negative-frequency components is called an *analytic signal*.[6] Therefore, in continuous time, every analytic signal $z(t)$ can be represented as

$$z(t) = \frac{1}{2\pi}\int_0^\infty Z(\omega)e^{j\omega t}d\omega$$

where $Z(\omega)$ is the complex coefficient (setting the amplitude and phase) of the positive-freqency complex sinusoid $\exp(j\omega t)$ at frequency ω.

Any sinusoid $A\cos(\omega t+\phi)$ in real life may be converted to a positive-frequency complex sinusoid $A\exp[j(\omega t+\phi)]$ by simply generating a phase-quadrature component $A\sin(\omega t + \phi)$ to serve as the "imaginary part":

$$Ae^{j(\omega t+\phi)} = A\cos(\omega t + \phi) + jA\sin(\omega t + \phi)$$

[6] In complex variables, "analytic" just means differentiable of all orders. Therefore, one would expect an "analytic signal" to simply be any signal which is differentiable of all orders at any point in time, i.e., one that admits a fully valid Taylor expansion about any point in time. However, *all* bandimited signals (being sums of finite-frequency sinusoids) are analytic in the complex-variables sense. Therefore, the signal processing term "analytic signal" is somewhat of a misnomer. It is included in this chapter only because it is a commonly used term in engineering practice.

The phase-quadrature component can be generated from the in-phase component by a simple quarter-cycle time shift.[7]

For more complicated signals which are expressible as a sum of many sinusoids, a *filter* can be constructed which shifts each sinusoidal component by a quarter cycle. This is called a *Hilbert transform filter*. Let $\mathcal{H}_t\{x\}$ denote the output at time t of the Hilbert-transform filter applied to the signal $x(\cdot)$. Ideally, this filter has magnitude 1 at all frequencies and introduces a phase shift of $-\pi/2$ at each positive frequency and $+\pi/2$ at each negative frequency. When a real signal $x(t)$ and its Hilbert transform $y(t) = \mathcal{H}_t\{x\}$ are used to form a new complex signal $z(t) = x(t) + jy(t)$, the signal $z(t)$ is the (complex) *analytic signal* corresponding to the real signal $x(t)$. In other words, for any real signal $x(t)$, the corresponding analytic signal $z(t) = x(t) + j\mathcal{H}_t\{x\}$ has the property that all "negative frequencies" of $x(t)$ have been "filtered out."

To see how this works, recall that these phase shifts can be impressed on a complex sinusoid by multiplying it by $\exp(\pm j\pi/2) = \pm j$. Consider the positive and negative frequency components at the particular frequency ω_0:

$$\begin{aligned} x_+(t) &\stackrel{\Delta}{=} e^{j\omega_0 t} \\ x_-(t) &\stackrel{\Delta}{=} e^{-j\omega_0 t} \end{aligned}$$

Now let's apply a -90 degrees phase shift to the positive-frequency component, and a $+90$ degrees phase shift to the negative-frequency component:

$$\begin{aligned} y_+(t) &= e^{-j\pi/2} e^{j\omega_0 t} = -j e^{j\omega_0 t} \\ y_-(t) &= e^{j\pi/2} e^{-j\omega_0 t} = j e^{-j\omega_0 t} \end{aligned}$$

Adding them together gives

$$\begin{aligned} z_+(t) &\stackrel{\Delta}{=} x_+(t) + jy_+(t) = e^{j\omega_0 t} - j^2 e^{j\omega_0 t} = 2e^{j\omega_0 t} \\ z_-(t) &\stackrel{\Delta}{=} x_-(t) + jy_-(t) = e^{-j\omega_0 t} + j^2 e^{-j\omega_0 t} = 0 \end{aligned}$$

[7] This operation is actually used in some real-world AM and FM radio receivers (particularly in digital radio receivers). The signal comes in centered about a high "carrier frequency" (such as 101 MHz for radio station FM 101), so it looks very much like a sinusoid at frequency 101 MHz. (The frequency modulation only varies the carrier frequency in a relatively tiny interval about 101 MHz. The total FM bandwidth including all the FM "sidebands" is about 100 kHz. AM bands are only 10kHz wide.) By delaying the signal by 1/4 cycle, a good approximation to the imaginary part of the analytic signal is created, and its instantaneous amplitude and frequency are then simple to compute from the analytic signal.

and sure enough, the negative frequency component is filtered out. (There is also a gain of 2 at positive frequencies which we can remove by defining the Hilbert transform filter to have magnitude 1/2 at all frequencies.)

For a concrete example, let's start with the real sinusoid

$$x(t) = 2\cos(\omega_0 t) = \exp(j\omega_0 t) + \exp(-j\omega_0 t).$$

Applying the ideal phase shifts, the Hilbert transform is

$$\begin{aligned} y(t) &= \exp(j\omega_0 t - j\pi/2) + \exp(-j\omega_0 t + j\pi/2) \\ &= -j\exp(j\omega_0 t) + j\exp(-j\omega_0 t) = 2\sin(\omega_0 t) \end{aligned}$$

The analytic signal is then

$$z(t) = x(t) + jy(t) = 2\cos(\omega_0 t) + j2\sin(\omega_0 t) = 2e^{j\omega_0 t},$$

by Euler's identity. Thus, in the sum $x(t) + jy(t)$, the negative-frequency components of $x(t)$ and $jy(t)$ cancel out in the sum, leaving only the positive-frequency component. This happens for any real signal $x(t)$, not just for sinusoids as in our example.

Figure 5.8 illustrates what is going on in the frequency domain. While we haven't "had" Fourier analysis yet, it should come as no surprise that the spectrum of a complex sinusoid $\exp(j\omega_0 t)$ will consist of a single "spike" at the frequency $\omega = \omega_0$ and zero at all other frequencies. (Just follow things intuitively for now, and revisit Fig. 5.8 after we've developed the Fourier theorems.) From the identity $2\cos(\omega_0 t) = \exp(j\omega_0 t) + \exp(-j\omega_0 t)$, we see that the spectrum contains unit-amplitude "spikes" at $\omega = \omega_0$ and $\omega = -\omega_0$. Similarly, the identity $2\sin(\omega_0 t) = [\exp(j\omega_0 t) - \exp(-j\omega_0 t)]/j = -j\exp(j\omega_0 t) + j\exp(-j\omega_0 t)$ says that we have an amplitude $-j$ spike at $\omega = \omega_0$ and an amplitude $+j$ spike at $\omega = -\omega_0$. Multiplying $y(t)$ by j results in $j\sin(\omega_0 t) = \exp(j\omega_0 t) - \exp(-j\omega_0 t)$ which is a unit-amplitude "up spike" at $\omega = \omega_0$ and a unit "down spike" at $\omega = -\omega_0$. Finally, adding together the first and third plots, corresponding to $z(t) = x(t) + jy(t)$, we see that the two up-spikes *add in phase* to give an amplitude 2 up-spike (which is $2\exp(j\omega_0 t)$), and the negative-frequency up-spike in the cosine is *canceled* by the down-spike in j times sine at frequency $-\omega_0$. This sequence of operations illustrates how the negative-frequency component $\exp(-j\omega_0 t)$ gets *filtered out* by the addition of $2\cos(\omega_0 t)$ and $j2\sin(\omega_0 t)$.

As a final example (and application), let $x(t) = A(t)\cos(\omega t)$, where $A(t)$ is a slowly varying amplitude envelope (slow compared with ω). This

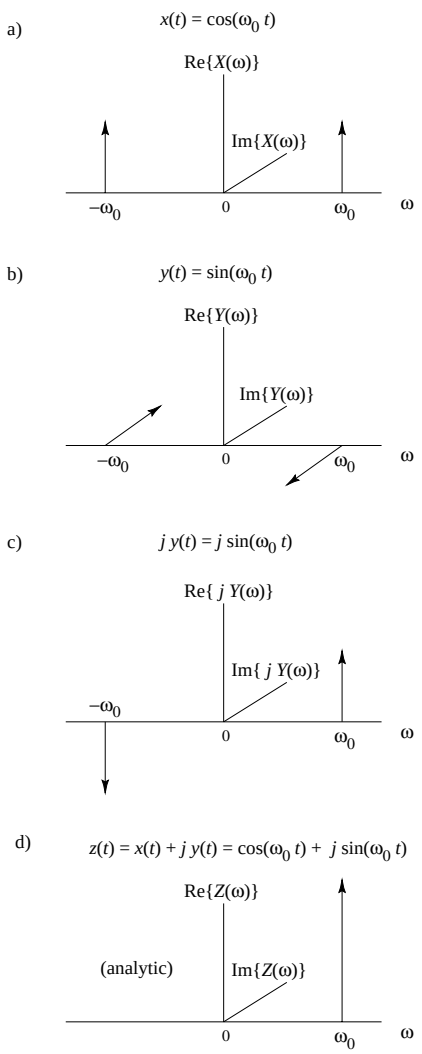

Figure 5.8: Creation of the analytic signal $z(t) = e^{j\omega_0 t}$ from the real sinusoid $x(t) = \cos(\omega_0 t)$ and the derived phase-quadrature sinusoid $y(t) = \sin(\omega_0 t)$, viewed in the frequency domain. a) Spectrum of x. b) Spectrum of y. c) Spectrum of jy. d) Spectrum of $z = x + jy$.

is an example of *amplitude modulation* applied to a sinusoid at "carrier frequency" ω (which is where you tune your AM radio). The Hilbert transform is almost exactly $y(t) \approx A(t)\sin(\omega t)$[8], and the analytic signal is $z(t) \approx A(t)e^{j\omega t}$. Note that AM *demodulation*[9] is now nothing more than the *absolute value*. I.e., $A(t) = |z(t)|$. Due to this simplicity, Hilbert transforms are sometimes used in making *amplitude envelope followers* for narrowband signals (i.e., signals with all energy centered about a single "carrier" frequency). AM demodulation is one application of a narrowband envelope follower.

5.3.5 Generalized Complex Sinusoids

We have defined sinusoids and extended the definition to include complex sinusoids. We now extend one more step by allowing for *exponential amplitude envelopes*:

$$y(t) \stackrel{\Delta}{=} \mathcal{A}e^{st}$$

where $\mathcal{A}$ and s are *complex*, and further defined as

$$\begin{aligned} \mathcal{A} &= Ae^{j\phi} \\ s &= \sigma + j\omega \end{aligned}$$

When $\sigma = 0$, we obtain

$$y(t) \stackrel{\Delta}{=} \mathcal{A}e^{j\omega t} = Ae^{j\phi}e^{j\omega t} = Ae^{j(\omega t+\phi)}$$

which is the complex sinusoid at amplitude A, radian frequency ω, and phase ϕ.

More generally, we have

$$\begin{aligned} y(t) &\stackrel{\Delta}{=} \mathcal{A}e^{st} \\ &\stackrel{\Delta}{=} Ae^{j\phi}e^{(\sigma+j\omega)t} \\ &= Ae^{(\sigma+j\omega)t+j\phi} \\ &= Ae^{\sigma t}e^{j(\omega t+\phi)} \\ &= Ae^{\sigma t}\left[\cos(\omega t+\phi) + j\sin(\omega t+\phi)\right] \end{aligned}$$

[8] If $A(t)$ were constant, this would be exact.

[9] *Demodulation* is the process of recovering the modulation signal. For amplitude modulation (AM), the modulated signal is of the form $y(t) = A(t)\cos(\omega_c t)$, where ω_c is the "carrier frequency", $A(t) = [1+\mu x(t)] \geq 0$ is the amplitude envelope (modulation), $x(t)$ is the modulation signal we wish to recover (the audio signal being broadcast in the case of AM radio), and μ is the modulation index for AM.

Defining $\tau = -1/\sigma$, we see that the generalized complex sinusoid is just the complex sinusoid we had before with an *exponential envelope*:

$$\begin{aligned} \mathrm{re}\left\{y(t)\right\} &= Ae^{-t/\tau}\cos(\omega t + \phi) \\ \mathrm{im}\left\{y(t)\right\} &= Ae^{-t/\tau}\sin(\omega t + \phi) \end{aligned}$$

5.3.6 Sampled Sinusoids

In discrete-time audio processing, such as we must do on a computer, we work with *samples* of continuous-time signals. Let f_s denote the sampling rate in Hz. For audio, we typically have $f_s > 40$ kHz, since the audio band nominally extends to 20 kHz. For compact discs (CDs), $f_s = 44.1$ kHz (or very close to that—I once saw Sony device using a sampling rate of $44,025$ Hz), while for digital audio tape (DAT), $f_s = 48$ kHz.

Let $T \stackrel{\Delta}{=} 1/f_s$ denote the *sampling period* in seconds. Then to convert from continuous to discrete time, we replace t by nT, where n is an integer interpreted as the *sample number*.

The sampled generalized complex sinusoid (which includes all other cases) is then

$$\begin{aligned} y(nT) &= \mathcal{A}e^{snT} = \mathcal{A}\left[e^{sT}\right]^n \\ &\stackrel{\Delta}{=} Ae^{j\phi}e^{(\sigma+j\omega)nT} \\ &= Ae^{\sigma nT}\left[\cos(\omega nT + \phi) + j\sin(\omega nT + \phi)\right] \\ &= A\left[e^{\sigma T}\right]^n\left[\cos(\omega nT + \phi) + j\sin(\omega nT + \phi)\right] \end{aligned}$$

5.3.7 Powers of z

Choose any two complex numbers z_0 and z_1, and form the sequence

$$x(n) \stackrel{\Delta}{=} z_0 z_1^n, \quad n = 0, 1, 2, 3, \ldots \tag{5.1}$$

What are the properties of this signal? Expressing the two complex numbers as

$$\begin{aligned} z_0 &= Ae^{j\phi} \\ z_1 &= e^{sT} = e^{(\sigma+j\omega)T} \end{aligned}$$

we see that the signal $x(n)$ is always a discrete-time generalized complex sinusoid, i.e., an exponentially enveloped complex sinusoid.

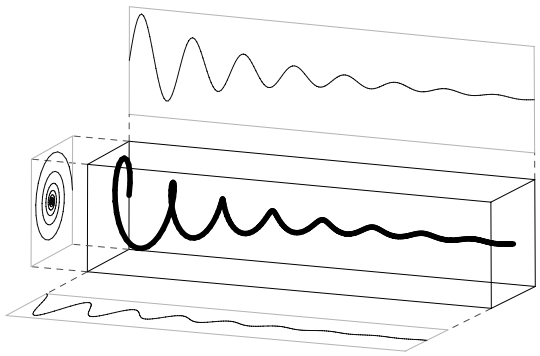

Figure 5.9: Exponentially decaying complex sinusoid and its projections.

Figure 5.9 shows a plot of a generalized (exponentially decaying) complex sinusoid versus time.

Note that the left projection (onto the z plane) is a decaying spiral, the lower projection (real-part vs. time) is an exponentially decaying cosine, and the upper projection (imaginary-part vs. time) is an exponentially enveloped sine wave.

5.3.8 Phasor & Carrier Components of Complex Sinusoids

If we restrict z_1 in Eq. (5.1) to have unit modulus, then we obtain a discrete-time *complex sinusoid*.

$$x(n) \stackrel{\Delta}{=} z_0 z_1^n = \left(Ae^{j\phi}\right) e^{j\omega nT} = Ae^{j(\omega nT+\phi)}, \quad n = 0, 1, 2, 3, \ldots \tag{5.2}$$

where we have defined

$$\begin{aligned} z_0 &\stackrel{\Delta}{=} Ae^{j\phi}, \quad \text{and} \\ z_1 &\stackrel{\Delta}{=} e^{j\omega T}. \end{aligned}$$

It is common terminology to call $z_0 = Ae^{j\phi}$ the sinusoidal *phasor*, and $z_1^n = e^{j\omega nT}$ the sinusoidal *carrier*.

For a *real* sinusoid

$$x_r(n) \stackrel{\Delta}{=} A\cos(\omega nT + \phi)$$

the phasor is again defined as $z_0 = Ae^{j\phi}$ and the carrier is $z_1^n = e^{j\omega nT}$. However, in this case, the real sinusoid is recovered from its complex sinusoid counterpart by taking the real part:

$$x_r(n) = \mathrm{re}\{z_0 z_1^n\}$$

The *phasor magnitude* $|z_0| = A$ is the *amplitude* of the sinusoid. The *phasor angle* $\angle z_0 = \phi$ is the *phase* of the sinusoid.

When working with complex sinusoids, as in Eq. (5.2), the phasor representation of a sinusoid can be thought of as simply the *complex amplitude* of the sinusoid $Ae^{j\phi}$. I.e., it is the complex constant that multiplies the carrier term $e^{j\omega nT}$.

Why Phasors are Important

LTI systems perform only four operations on a signal: copying, scaling, delaying, and adding. As a result, each output is always a *linear combination* of delayed copies of the input signal(s). (A *linear combination* is simply a weighted sum.) In any linear combination of delayed copies of a complex sinusoid

$$y(n) = \sum_{i=1}^{N} g_i x(n - d_i)$$

where g_i is a weighting factor, d_i is the ith delay, and

$$x(n) = e^{j\omega nT}$$

is a complex sinusoid, the "carrier term" $e^{j\omega nT}$ can be "factored out" of the linear combination:

$$\begin{aligned} y(n) &= \sum_{i=1}^{N} g_i e^{j[\omega(n-d_i)T]} = \sum_{i=1}^{N} g_i e^{j\omega nT} e^{-j\omega d_i T} \\ &= e^{j\omega nT} \sum_{i=1}^{N} g_i e^{-j\omega d_i T} = x(n) \sum_{i=1}^{N} g_i e^{-j\omega d_i T} \end{aligned}$$

The operation of the LTI system on a complex sinusoids is thus reduced to a calculation involving only phasors, which are simply complex numbers.

Since every signal can be expressed as a linear combination of complex sinusoids, this analysis can be applied to any signal by expanding the signal into its weighted sum of complex sinusoids (i.e., by expressing it as an inverse Fourier transform).

5.3.9 Why Generalized Complex Sinusoids are Important

As a preview of things to come, note that one signal $y(\cdot)$[10] is *projected* onto another signal $x(\cdot)$ using an *inner product*. The inner product $\langle y, x\rangle$ computes the *coefficient of projection*[11] of y onto x. If $x(n) = e^{j\omega_k nT}, n = 0, 1, 2, \ldots, N-1$ (a sampled, unit-amplitude, zero-phase, complex sinusoid), then the inner product computes the *Discrete Fourier Transform* (DFT), provided the frequencies are chosen to be $\omega_k = 2\pi k f_s/N$. For the DFT, the inner product is specifically

$$\langle y, x\rangle \triangleq \sum_{n=0}^{N-1} y(n)\overline{x(n)} = \sum_{n=0}^{N-1} y(n)e^{-j2\pi nk/N} \triangleq \mathrm{DFT}_k(y) \triangleq Y(\omega_k)$$

Another commonly used case is the *Discrete Time Fourier Transform* (DTFT) which is like the DFT, except that the transform accepts an infinite number of samples instead of only N. In this case, frequency is continuous, and

$$\langle y, x\rangle = \sum_{n=0}^{\infty} y(n)e^{-j\omega nT} \triangleq \mathrm{DTFT}_\omega(y)$$

The DTFT is what you get in the limit as the number of samples in the DFT approaches infinity. The lower limit of summation remains zero because we are assuming all signals are zero for negative time. This means we are working with *unilateral* Fourier transforms. There are also corresponding *bilateral* transforms for which the lower summation limit is $-\infty$.

If, more generally, $x(n) = z^n$ (a sampled complex sinusoid with exponential growth or decay), then the inner product becomes

$$\langle y, x\rangle = \sum_{n=0}^{\infty} y(n)z^{-n}$$

and this is the definition of the z *transform*. It is a generalization of the DTFT: The DTFT equals the z transform evaluated on the *unit circle* in the z plane. In principle, the z transform can also be recovered from the

[10] The notation $y(n)$ denotes a single *sample* of the signal y at sample n, while the notation $y(\cdot)$ or simply y denotes the *entire signal* for all time.

[11] The coefficient of projection of a signal y onto another signal x can be thought of as a measure of how much of x is present in y. We will consider this topic in some detail later on.

DTFT by means of "analytic continuation" from the unit circle to the entire z plane (subject to mathematical disclaimers which are unnecessary in practical applications since they are always finite).

Why have a z tranform when it seems to contain no more information than the DTFT? It is useful to generalize from the unit circle (where the DFT and DTFT live) to the entire complex plane (the z transform's domain) for a number of reasons. First, it allows transformation of *growing* functions of time such as unstable exponentials; the only limitation on growth is that it cannot be faster than exponential. Secondly, the z transform has a deeper algebraic structure over the complex plane as a whole than it does only over the unit circle. For example, the z transform of any finite signal is simply a *polynomial* in z. As such, it can be fully characterized (up to a constant scale factor) by its *zeros* in the z plane. Similarly, the z transform of an *exponential* can be characterized by a single point of the transform (the point which *generates* the exponential); since the z transform goes to infinity at that point, it is called a *pole* of the transform. More generally, the z transform of any *generalized complex sinusoid* is simply a *pole* located at the point which generates the sinusoid. Poles and zeros are used extensively in the analysis of *recursive digital filters*. On the most general level, every finite-order, linear, time-invariant, discrete-time system is fully specified (up to a scale factor) by its poles and zeros in the z plane.

In the *continuous-time* case, we have the *Fourier transform* which projects y onto the continuous-time sinusoids defined by $x(t) = e^{j\omega t}$, and the appropriate inner product is

$$\langle y, x\rangle = \int_0^\infty y(t)e^{-j\omega t}dt \stackrel{\Delta}{=} Y(\omega)$$

Finally, the *Laplace transform* is the continuous-time counterpart of the z transform, and it projects signals onto exponentially growing or decaying complex sinusoids:

$$\langle y, x\rangle = \int_0^\infty y(t)e^{-st}dt \stackrel{\Delta}{=} Y(s)$$

The Fourier transform equals the Laplace transform evaluated along the "$j\omega$ axis" in the s plane, i.e., along the line $s = j\omega$, for which $\sigma = 0$. Also, the Laplace transform is obtainable from the Fourier transform via analytic continuation. The usefulness of the Laplace transform relative to the Fourier transform is exactly analogous to that of the z transform outlined above.

5.3.10 Comparing Analog and Digital Complex Planes

In signal processing, it is customary to use s as the Laplace transform variable for continuous-time analysis, and z as the z-transform variable for discrete-time analysis. In other words, for continuous-time systems, the frequency domain is the "s plane", while for discrete-time systems, the frequency domain is the "z plane." However, both are simply complex planes.

Figure 5.10 illustrates the various sinusoids e^{st} represented by points in the s plane. The frequency axis is $s = j\omega$, called the "$j\omega$ axis," and points along it correspond to complex sinusoids, with dc at $s = 0$ ($e^{0t} = 1$). The upper-half plane corresponds to positive frequencies (counterclockwise circular or corkscrew motion) while the lower-half plane corresponds to negative frequencies (clockwise motion). In the left-half plane we have decaying (stable) exponential envelopes, while in the right-half plane we have growing (unstable) exponential envelopes. Along the real axis ($s = \sigma$), we have pure exponentials. Every point in the s plane can be said to correspond to some generalized complex sinusoids, $x(t) = \mathcal{A}e^{st}, t \geq 0$ with special cases being complex sinusoids $\mathcal{A}e^{j\omega t}$, exponentials $Ae^{\sigma t}$, and the constant function $x(t) = 1$ (dc).

Figure 5.11 shows examples of various sinusoids $z^n = [e^{sT}]^n$ represented by points in the z plane. The frequency axis is the "unit circle" $z = e^{j\omega T}$, and points along it correspond to *sampled* complex sinusoids, with dc at $z = 1$ ($1^n = [e^{0T}]^n = 1$). As in the s plane, the upper-half plane corresponds to positive frequencies while the lower-half plane corresponds to negative frequencies. Inside the unit circle, we have decaying (stable) exponential envelopes, while outside the unit circle, we have growing (unstable) exponential envelopes. Along the positive real axis ($z > 0, \mathrm{im}\{z\} = 0$), we have pure exponentials, but along the negative real axis ($z < 0, \mathrm{im}\{z\} = 0$), we have exponentially enveloped sampled sinusoids at frequency $f_s/2$ (exponentially enveloped alternating sequences). The negative real axis in the z plane is normally a place where all signal z transforms should be zero, and all system responses should be highly attenuated, since there should never be any energy at exactly half the sampling rate (where amplitude and phase are ambiguously linked). Every point in the z plane can be said to correspond to some sampled generalized complex sinusoids $x(n) = \mathcal{A}z^n = \mathcal{A}[e^{sT}]^n, n \geq 0$, with special cases being sampled complex sinusoids $\mathcal{A}e^{j\omega nT}$, exponentials $Ae^{\sigma nT}$, and the constant function $x = [1, 1, 1, \ldots]$ (dc).

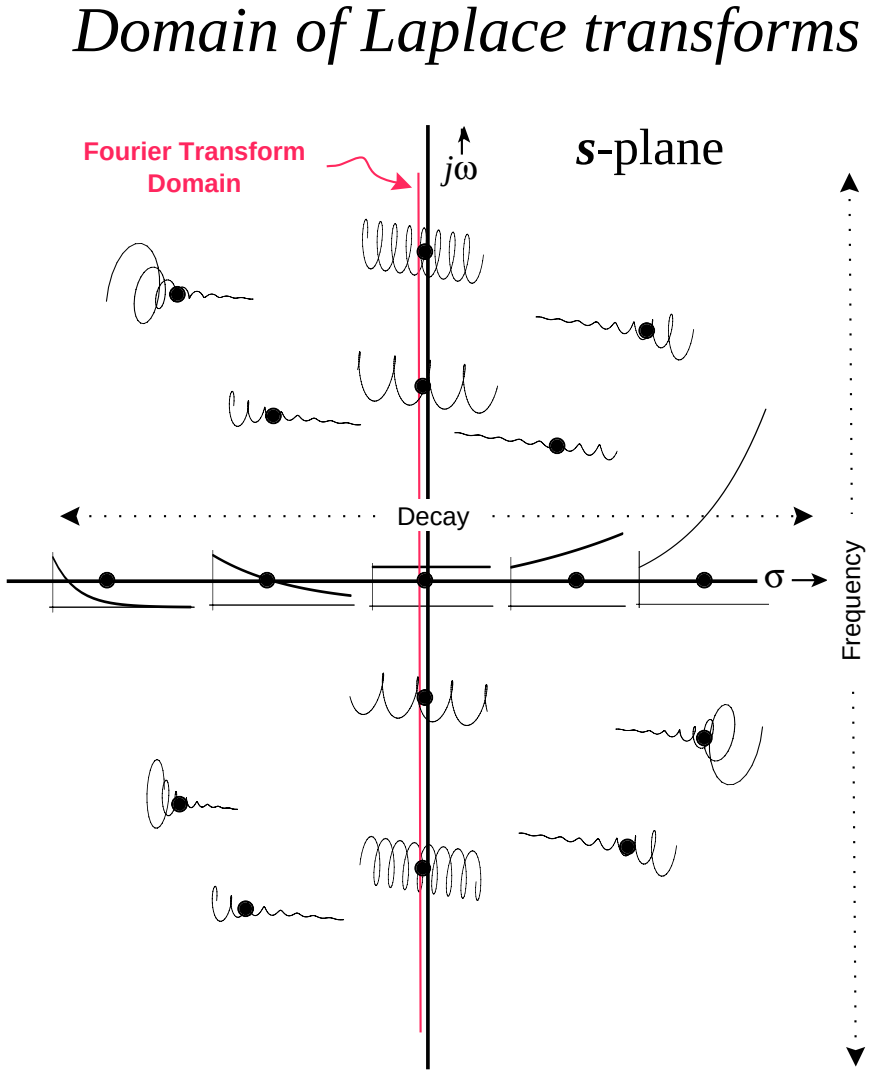

Figure 5.10: Generalized complex sinusoids represented by points in the s plane.

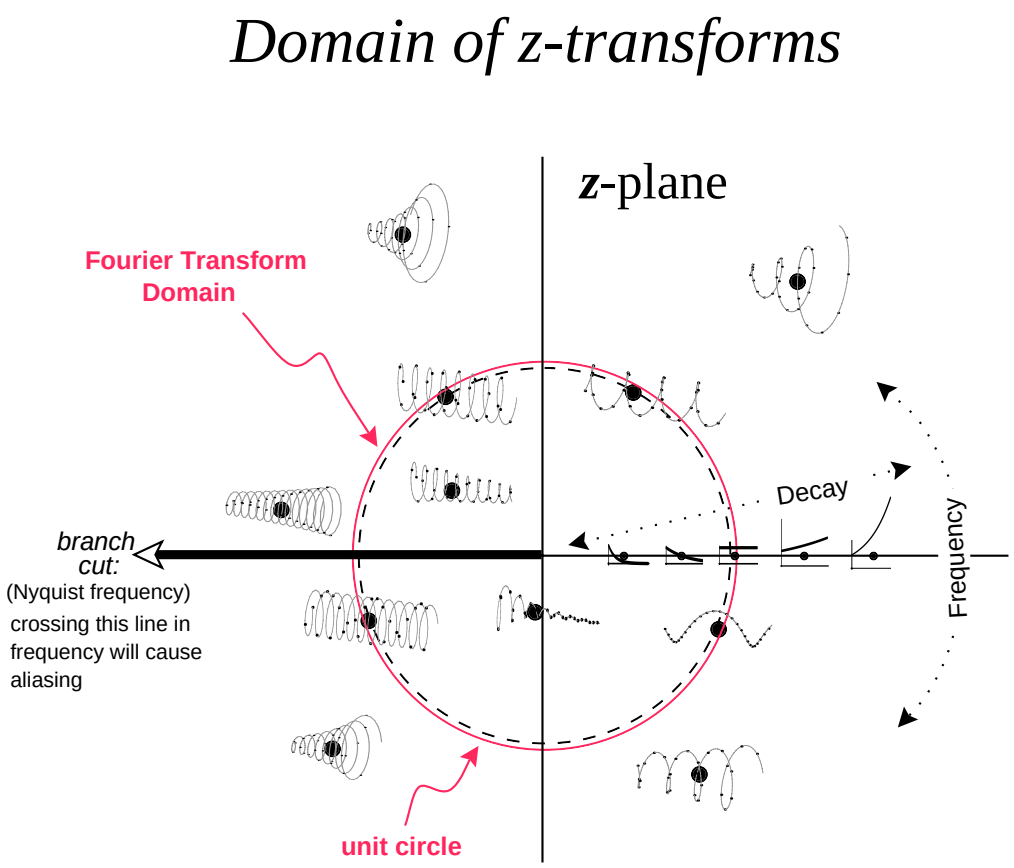

Figure 5.11: Generalized complex sinusoids represented by points in the z plane.

In summary, the exponentially enveloped ("generalized") complex sinusoid is the fundamental signal upon which other signals are "projected" in order to compute a Laplace transform in the continuous-time case, or a z transform in the discrete-time case. As a special case, if the exponential envelope is eliminated (set to 1), leaving only a complex sinusoid, then the projection reduces to the Fourier transform in the continuous-time case, and either the DFT (finite length) or DTFT (infinite length) in the discrete-time case. Finally, there are still other variations, such as short-time Fourier transforms (STFT) and wavelet transforms, which utilize further modifications such as projecting onto *windowed* complex sinusoids. Music 420[12] delves into these topics.

5.4 Mathematica for Selected Plots

The Mathematica code for producing Fig. 5.1 (minus the annotations which were done using NeXT Draw and EquationBuilder from Lighthouse Design) is

```
Plot[10 Sin[2 Pi 2.5 t + Pi/4],{t,0,1},
    PlotLabel->"10 Sin[2 Pi 2.5 t + Pi/4]",
    PlotPoints->500,
    AxesLabel->{" Sec", "Amp."}];
```

The Mathematica code for Fig. 5.2 is

```
Show[
    Plot[Sin[2 Pi 2.5 t],{t,-0.1,1.1},
        PlotPoints->500,
        AxesLabel->{`` Time (Sec)'', ``Amplitude''}],
    Plot[Cos[2 Pi 2.5 t],{t,-0.1,1.1},
        PlotPoints->500,
        PlotStyle->Dashing[{0.01,0.01}]
];
```

For the complex sinusoid plots (Fig. 5.7 and Fig. 5.9), see the Mathematica notebook `ComplexSinusoid.nb`[13] on Craig Sapp's web page[14] of Mathematica notebooks. (The package `SCMTheory.m`[15] is required.)

[12] http://www-ccrma.stanford.edu/CCRMA/Courses/420/Welcome.html
[13] http://www-ccrma.stanford.edu/CCRMA/Software/SCMP/SCMTheory/ComplexSinusoid.nb.gz
[14] http://www-ccrma/CCRMA/Software/SCMP/
[15] http://www-ccrma.stanford.edu/CCRMA/Software/SCMP/SCMTheory/SCMTheory.m

5.5 Acknowledgement

Many thanks to Craig Stuart Sapp <craig@ccrma.stanford.edu> for contributing the Mathematica magic for Figures 5.7, 5.9, 5.10, and 5.11.

Chapter 6

Geometric Signal Theory

This chapter provides an introduction to the elements of geometric signal theory, including vector spaces, norms, inner products, orthogonality, projection of one signal onto another, and elementary vector space operations.

6.1 The DFT

For a length N complex sequence $x(n)$, $n = 0, 1, 2, \ldots, N-1$, the *discrete Fourier transform* (DFT) is defined by

$$X(\omega_k) \triangleq \sum_{n=0}^{N-1} x(n) e^{-j\omega_k t_n} = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}, \quad k = 0, 1, 2, \ldots N-1$$

$$\begin{aligned} t_n &\triangleq nT = n\text{th sampling instant (sec)} \\ \omega_k &\triangleq k\Omega = k\text{th frequency sample (rad/sec)} \\ T &\triangleq 1/f_s = \text{time sampling interval (sec)} \\ \Omega &\triangleq 2\pi f_s/N = \text{frequency sampling interval (sec)} \end{aligned}$$

We are now in a position to have a full understanding of the transform *kernel*:

$$e^{-j\omega_k t_n} = \cos(\omega_k t_n) - j \sin(\omega_k t_n)$$

The kernel consists of samples of a complex sinusoid at N discrete frequencies ω_k uniformly spaced between 0 and the sampling rate $\omega_s \triangleq 2\pi f_s$.

All that remains is to understand the purpose and function of the summation over n of the pointwise product of $x(n)$ times each complex sinusoid. We will learn that this can be interpreted as an *inner product* operation which computes the *coefficient of projection* of the signal x onto the complex sinusoid $\cos(\omega_k t_n) + j\sin(\omega_k t_n)$. As such, $X(\omega_k)$, the DFT at frequency ω_k, is a measure of the amplitude and phase of the complex sinusoid at that frequency which is present in the input signal x. This is the basic function of all transform summations (in discrete time) and integrals (in continuous time) and their kernels.

6.2 Signals as Vectors

For the DFT, all signals and spectra are length N. A length N sequence x can be denoted by $x(n)$, $n = 0, 1, 2, \ldots N-1$, where $x(n)$ may be real ($x \in \mathbf{R}^N$) or complex ($x \in \mathbf{C}^N$). We now wish to regard x as a *vector* $\underline{x}$[1] in an N dimensional *vector space*. That is, each sample $x(n)$ is regarded as a *coordinate* in that space. A *vector* $\underline{x}$ is mathematically a single *point* in N-space represented by a list of coordinates $(x_0, x_1, x_2, \ldots, x_{N-1})$ called an *N-tuple*. (The notation x_n means the same thing as $x(n)$.) It can be interpreted geometrically as an arrow in N-space from the origin $\underline{0} \stackrel{\Delta}{=} (0, 0, \ldots, 0)$ to the point $\underline{x} \stackrel{\Delta}{=} (x_0, x_1, x_2, \ldots, x_{N-1})$.

We define the following as equivalent:

$$x \stackrel{\Delta}{=} \underline{x} \stackrel{\Delta}{=} x(\cdot) \stackrel{\Delta}{=} (x_0, x_1, \ldots, x_{N-1}) \stackrel{\Delta}{=} [x_0, x_1, \ldots, x_{N-1}] \stackrel{\Delta}{=} [x_0\ x_1\ \cdots\ x_{N-1}]$$

where $x_n \stackrel{\Delta}{=} x(n)$ is the nth sample of the signal (vector) x. From now on, unless specifically mentioned otherwise, *all signals are length N*.

An Example Vector View: $N = 2$

Consider the example two-sample signal $x = (2, 3)$ graphed in Fig. 6.1.

Under the geometric interpretation of a length N signal, each sample is a *coordinate* in the N dimensional space. Signals which are only two samples long are not terribly interesting to hear, but they are easy to plot geometrically.

[1] We'll use an underline to emphasize the vector interpretation, but there is no difference between x and $\underline{x}$. For purposes of this course, a *signal* is the same thing as a vector.

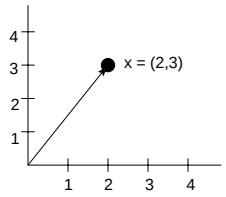

Figure 6.1: A length 2 signal $x = (2, 3)$ plotted as a vector in 2D space.

6.3 Vector Addition

Given two vectors in $\mathbf{R}^N$, say $\underline{x} = (x_0, x_1, \ldots, x_{N-1})$ and $\underline{y} = (y_0, y_1, \ldots, y_{N-1})$, the *vector sum* is defined by *elementwise* addition. If we denote the sum by $w \stackrel{\Delta}{=} x + y$, then we have $w(n) = x(n) + y(n)$ for $n = 0, 1, 2, \ldots, N-1$.

The vector diagram for the sum of two vectors can be found using the parallelogram rule, as shown in Fig. 6.2 for $N = 2$, $x = (2, 3)$, and $y = (4, 1)$.

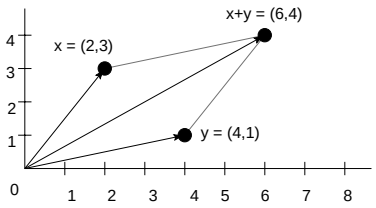

Figure 6.2: Geometric interpretation of a length 2 vector sum.

Also shown are the lighter construction lines which complete the parallelogram started by x and y, indicating where the endpoint of the sum $x + y$ lies. Since it is a parallelogram, the two construction lines are congruent to the vectors x and y. As a result, the vector sum is often expressed as a *triangle* by translating the origin of one member of the sum to the tip of the other, as shown in Fig. 6.3.

In the figure, x was translated to the tip of y. It is equally valid to translate y to the tip of x, because vector addition is *commutative*, i.e., $x + y = y + x$.

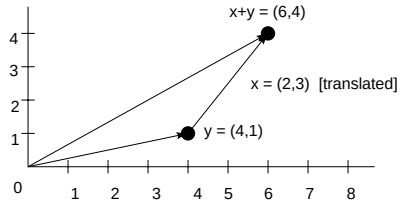

Figure 6.3: Vector sum with translation of one vector to the tip of the other.

6.4 Vector Subtraction

Figure 6.7 illustrates the vector difference $w = x - y$ between $x = (2, 3)$ and $y = (4, 1)$. From the coordinates, we compute $w = x - y = (-2, 2)$.

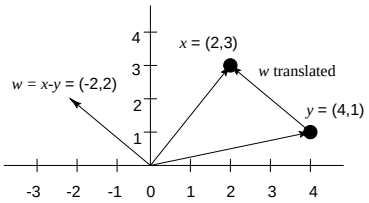

Figure 6.4: Geometric interpretation a difference vector.

Note that the difference vector w may be drawn from the tip of y to the tip of x rather than from the origin to the point $(-2, 2)$; this is a customary practice which emphasizes relationships among vectors, but the translation in the plot has no effect on the mathematical definition or properties of the vector. Subtraction, however, is not commutative.

To ascertain the proper orientation of the difference vector $w = x - y$, rewrite its definition as $x = y + w$, and then it is clear that the vector x should be the sum of vectors y and w, hence the arrowhead is on the correct endpoint.

6.5 Signal Metrics

This section defines some useful functions of signals.

The *mean* of a signal x (more precisely the "sample mean") is defined

as its *average value*:

$$\mu_x \stackrel{\Delta}{=} \frac{1}{N}\sum_{n=0}^{N-1} x_n \qquad \text{(mean of } x\text{)}$$

The *total energy* of a signal x is defined the *sum of squared moduli*:

$$\mathcal{E}_x \stackrel{\Delta}{=} \sum_{n=0}^{N-1} |x_n|^2 \qquad \text{(energy of } x\text{)}$$

Energy is the "ability to do work." In physics, energy and work are in units of "force times distance," "mass times velocity squared," or other equivalent combinations of units. The energy of a *pressure wave* is the integral over time of the squared pressure divided by the wave impedance the wave is traveling in. The energy of a *velocity wave* is the integral over time of the squared velocity times the wave impedance. In audio work, a signal x is typically a list of *pressure samples* derived from a microphone signal, or it might be samples of *force* from a piezoelectric transducer, *velocity* from a magnetic guitar pickup, and so on. In all of these cases, the total physical energy associated with the signal is proportional to the sum of squared signal samples. (Physical connections in signal processing are explored more deeply in Music 421[2].)

The *average power* of a signal x is defined the *energy per sample*:

$$\mathcal{P}_x \stackrel{\Delta}{=} \frac{\mathcal{E}_x}{N} = \frac{1}{N}\sum_{n=0}^{N-1} |x_n|^2 \qquad \text{(average power of } x\text{)}$$

Another common description when x is real is the "mean square." When x is a complex sinusoid, i.e., $x(n) = Ae^{j(\omega nT+\phi)}$, then $\mathcal{P}_x = A^2$; in other words, for complex sinusoids, the average power equals the *instantaneous power* which is the amplitude squared.

Power is always in physical units of energy per unit time. It therefore makes sense to define the average signal power as the total signal energy divided by its length. We normally work with signals which are functions of time. However, if the signal happens instead to be a function of distance (e.g., samples of displacement along a vibrating string), then the "power" as defined here still has the interpretation of a *spatial energy density*. Power, in contrast, is a temporal energy density.

[2] http://www-ccrma.stanford.edu/CCRMA/Courses/421/

The *root mean square* (RMS) level of a signal x is simply $\sqrt{\mathcal{P}_x}$. However, note that in practice (especially in audio work) an RMS level may be computed after subtracting out the mean value. Here, we call that the *variance*.

The *variance* (more precisely the *sample variance*) of the signal x is defined as the power of the signal with its sample mean removed:

$$\sigma_x^2 \stackrel{\Delta}{=} \frac{1}{N}\sum_{n=0}^{N-1} |x_n - \mu_x|^2 \qquad \text{(variance of } x\text{)}$$

It is quick to show that, for real signals, we have

$$\sigma_x^2 = \mathcal{P}_x - \mu_x^2$$

which is the "mean square minus the mean squared." We think of the variance as the power of the non-constant signal components (i.e., everything but dc). The terms "sample mean" and "sample variance" come from the field of *statistics*, particularly the theory of *stochastic processes*. The field of *statistical signal processing* [12] is firmly rooted in statistical topics such as "probability," "random variables," "stochastic processes," and "time series analysis." In this reader, we will only touch lightly on a few elements of statistical signal processing in a self-contained way.

The *norm* of a signal x is defined as the square root of its total energy:

$$\|x\| \stackrel{\Delta}{=} \sqrt{\mathcal{E}_x} = \sqrt{\sum_{n=0}^{N-1} |x_n|^2} \qquad \text{(norm of } x\text{)}$$

We think of $\|x\|$ as the *length* of x in N-space. Furthermore, $\|x-y\|$ is regarded as the *distance* between x and y. The norm can also be thought of as the "absolute value" or "radius" of a vector.[3]

Example: Going back to our simple 2D example $x = [2, 3]$, we can compute its norm as $\|x\| = \sqrt{2^2 + 3^2} = \sqrt{13}$. The physical interpretation of the norm as a distance measure is shown in Fig. 6.5.

Example: Let's also look again at the vector-sum example, redrawn in Fig. 6.6.

The norm of the vector sum $w = x + y$ is

$$\|w\| \stackrel{\Delta}{=} \|x+y\| \stackrel{\Delta}{=} \|(2,3)+(4,1)\| = \|(6,4)\| = \sqrt{6^2+4^2} = \sqrt{52} = 2\sqrt{13}$$

[3]You might wonder why the norm of $\underline{x}$ is not written as $|\underline{x}|$. There would be no problem with this since $|\underline{x}|$ is undefined. However, the historically adopted notation is instead $\|\underline{x}\|$.

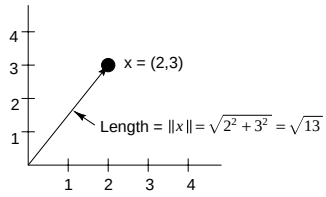

Figure 6.5: Geometric interpretation of a signal norm in 2D.

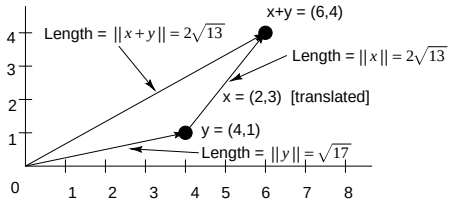

Figure 6.6: Length of vectors in sum.

while the norms of x and y are $\sqrt{13}$ and $\sqrt{17}$, respectively. We find that $\|x+y\| < \|x\|+\|y\|$ which is an example of the *triangle inequality*. (Equality occurs only when x and y are colinear, as can be seen geometrically from studying Fig. 6.6.)

Example: Consider the vector-difference example diagrammed in Fig. 6.7.

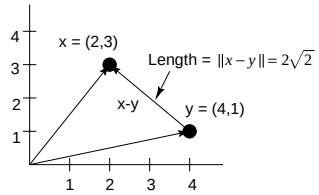

Figure 6.7: Length of a difference vector.

The norm of the difference vector $w = x - y$ is

$$\|w\| \stackrel{\Delta}{=} \|x-y\| \stackrel{\Delta}{=} \|(2,3)-(4,1)\| = \|(-2,2)\| = \sqrt{(-2)^2+(2)^2} = 2\sqrt{2}$$

Other Norms

Since our main norm is the square root of a sum of squares, we are using what is called an L_2 *norm* and we may write $\|x\|_2$ to emphasize this fact.

We could equally well have chosen a *normalized L_2 norm*:

$$\|x\|_{\tilde{2}} \stackrel{\Delta}{=} \sqrt{\mathcal{P}_x} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}|x_n|^2} \qquad \text{(normalized } L_2 \text{ norm of } x)$$

which is simply the "RMS level" of x.

More generally, the *L_p norm* of $x \in \mathbf{C}^N$ is defined

$$\|x\|_p \stackrel{\Delta}{=} \left(\sum_{n=0}^{N-1}|x_n|^p\right)^{1/p}$$

The most interesting L_p norms are

- $p = 1$: The L_1, "absolute value," or "city block" norm.
- $p = 2$: The L_2, "Euclidean," "root energy," or "least squares" norm.
- $p = \infty$: The L_∞, "Chebyshev," "supremum," "minimax," or "uniform" norm.

Note that the case $p = \infty$ is a limiting case which becomes

$$\|x\|_\infty = \max_{0\le n<N}|x_n|$$

There are many other possible choices of norm. To qualify as a norm on $\mathbf{C}^N$, a real-valued signal function $f(\underline{x})$ must satisfy the following three properties:

1. $f(\underline{x}) = \underline{0} \Leftrightarrow \underline{x} = 0$
2. $f(\underline{x} + \underline{y}) \le f(\underline{x}) + f(\underline{y})$
3. $f(c\underline{x}) = |c|\, f(\underline{x}),\ \forall c \in \mathcal{C}$

The first property, "positivity," says only the zero vector has norm zero. The second property is "subadditivity" and is sometimes called the "triangle inequality" for reasons which can be seen by studying Fig. 6.3. The third property says the norm is "absolutely homogeneous" with respect to scalar multiplication (which can be complex, in which case the phase of the scalar has no effect).

Mathematically, what we are working with so far is called a *Banach space* which a *normed linear vector space*. To summarize, we defined our

vectors as any list of N real or complex numbers which we interpret as coordinates in the N-dimensional vector space. We also defined vector addition in the obvious way. It turns out we have to also define *scalar multiplication*, that is, multiplication of a vector by a scalar which we also take to be an element of the field of real or complex numbers. This is also done in the obvious way which is to multiply each coordinate of the vector by the scalar. To have a linear vector space, it must be *closed* under vector addition and scalar multiplication. That means given any two vectors $\underline{x} \in \mathbf{C}^N$ and $\underline{y} \in \mathbf{C}^N$ from the vector space, and given any two scalars $c_1 \in \mathcal{C}$ and $c_2 \in \mathcal{C}$ from the field of scalars, then any *linear combination* $c_1\underline{x} + c_2\underline{y}$ must also be in the space. Since we have used the field of complex numbers $\mathcal{C}$ (or real numbers $\mathcal{R}$) to define both our scalars and our vector components, we have the necessary closure properties so that any linear combination of vectors from $\mathbf{C}^N$ lies in $\mathbf{C}^N$. Finally, the definition of a norm (any norm) elevates a vector space to a Banach space.

6.6 The Inner Product

The *inner product* (or "dot product") is an operation on two vectors which produces a scalar. Adding an inner product to a Banach space produces a *Hilbert space* (or "inner product space"). There are many examples of Hilbert spaces, but we will only need $\{\mathbf{C}^N, \mathcal{C}\}$ for this reader (complex length N vectors and complex scalars).

The *inner product* between two (complex) N-vectors x and y is defined by

$$\langle x, y \rangle \stackrel{\Delta}{=} \sum_{n=0}^{N-1} x(n)\overline{y(n)}$$

The complex conjugation of the second vector is done in order that a *norm* will be *induced* by the inner product:

$$\langle x, x \rangle = \sum_{n=0}^{N-1} x(n)\overline{x(n)} = \sum_{n=0}^{N-1} |x(n)|^2 \stackrel{\Delta}{=} \mathcal{E}_x = \|x\|^2$$

As a result, the inner product is *conjugate symmetric*:

$$\langle y, x \rangle = \overline{\langle x, y \rangle}$$

Note that the inner product takes $\mathbf{C}^N \times \mathbf{C}^N$ to $\mathcal{C}$. That is, two length N complex vectors are mapped to a complex scalar.

Example: For $N = 3$ we have, in general,

$$\langle x, y\rangle = x_0\overline{y_0} + x_1\overline{y_1} + x_2\overline{y_2}$$

Let

$$\begin{aligned} x &= [0, j, 1] \\ y &= [1, j, j] \end{aligned}$$

Then

$$\langle x, y\rangle = 0 \cdot 1 + j \cdot (-j) + 1 \cdot (-j) = 0 + 1 + (-j) = 1 - j$$

6.6.1 Linearity of the Inner Product

Any function $f(x)$ of a vector $x \in \mathbf{C}^N$ (which we may call an *operator* on $\mathbf{C}^N$) is said to be *linear* if for all $x_1 \in \mathbf{C}^N$ and $x_2 \in \mathbf{C}^N$, and for all scalars c_1 and c_2 in $\mathcal{C}$, we have

$$f(c_1x_1 + c_2x_2) = c_1f(x_1) + c_2f(x_2)$$

A linear operator thus "commutes with mixing."

Linearity consists of two component properties,

- *additivity*: $f(x_1 + x_2) = f(x_1) + f(x_2)$, and
- *homogeneity*: $f(c_1x_1) = c_1f(x_1)$.

The inner product $\langle x, y\rangle$ is *linear* in its first argument, i.e.

$$\langle c_1x_1 + c_2x_2, y\rangle = c_1\langle x_1, y\rangle + c_2\langle x_2, y\rangle$$

This is easy to show from the definition:

$$\begin{aligned} \langle c_1x_1 + c_2x_2, y\rangle &\triangleq \sum_{n=0}^{N-1} [c_1x_1(n) + c_2x_2(n)]\,\overline{y(n)} \\ &= \sum_{n=0}^{N-1} c_1x_1(n)\overline{y(n)} + \sum_{n=0}^{N-1} c_2x_2(n)\overline{y(n)} \\ &= c_1\sum_{n=0}^{N-1} x_1(n)\overline{y(n)} + c_2\sum_{n=0}^{N-1} x_2(n)\overline{y(n)} \\ &\triangleq c_1\langle x_1, y\rangle + c_2\langle x_2, y\rangle \end{aligned}$$

The inner product is also *additive* in its second argument, i.e.,

$$\langle x, y_1 + y_2 \rangle = \langle x, y_1 \rangle + \langle x, y_2 \rangle$$

but it is only *conjugate homogeneous* in its second argument, since

$$\langle x, c_1 y_1 \rangle = \overline{c_1} \langle x, y_1 \rangle \neq c_1 \langle x, y_1 \rangle$$

The inner product *is* strictly linear in its second argument with respect to *real* scalars:

$$\langle x, r_1 y_1 + r_2 y_2 \rangle = r_1 \langle x, y_1 \rangle + r_2 \langle x, y_2 \rangle, \quad r_i \in \mathcal{R}$$

Since the inner product is linear in both of its arguments for real scalars, it is often called a *bilinear operator* in that context.

6.6.2 Norm Induced by the Inner Product

We may define a *norm* on $\underline{x} \in \mathbf{C}^N$ using the inner product:

$$\|\underline{x}\| \stackrel{\Delta}{=} \sqrt{\langle \underline{x}, \underline{x} \rangle}$$

It is straightforward to show that properties 1 and 3 of a norm hold. Property 2 follows easily from the Schwarz Inequality which is derived in the following subsection. Alternatively, we can simply observe that the inner product induces the well known L_2 norm on $\mathbf{C}^N$.

6.6.3 Cauchy-Schwarz Inequality

The *Cauchy-Schwarz Inequality* (or "Schwarz Inequality") states that for all $\underline{x} \in \mathbf{C}^N$ and $\underline{y} \in \mathbf{C}^N$, we have

$$\left| \langle \underline{x}, \underline{y} \rangle \right| \leq \|\underline{x}\| \cdot \|\underline{y}\|$$

with equality if and only if $\underline{x} = c\underline{y}$ for some scalar c.

We can quickly show this for real vectors $\underline{x} \in \mathbf{R}^N$, $\underline{y} \in \mathbf{R}^N$, as follows: If either $\underline{x}$ or $\underline{y}$ is zero, the inequality holds (as equality). Assuming both are nonzero, let's scale them to unit-length by defining the normalized vectors $\tilde{x} \stackrel{\Delta}{=} \underline{x}/\|\underline{x}\|$, $\tilde{y} \stackrel{\Delta}{=} \underline{y}/\|\underline{y}\|$, which are unit-length vectors lying on the

"unit ball" in $\mathbf{R}^N$ (a hypersphere of radius 1). We have

$$\begin{aligned}
0 \leq \|\tilde{x}-\tilde{y}\|^2 &= \langle \tilde{x}-\tilde{y}, \tilde{x}-\tilde{y}\rangle \\
&= \langle \tilde{x},\tilde{x}\rangle - \langle \tilde{x},\tilde{y}\rangle - \langle \tilde{y},\tilde{x}\rangle + \langle \tilde{y},\tilde{y}\rangle \\
&= \|\tilde{x}\|^2 - \left[\langle \tilde{x},\tilde{y}\rangle + \overline{\langle \tilde{x},\tilde{y}\rangle}\right] + \|\tilde{y}\|^2 \\
&= 2 - 2\mathrm{re}\left\{\langle \tilde{x},\tilde{y}\rangle\right\} \\
&= 2 - 2\left\langle \tilde{x},\tilde{y}\right\rangle
\end{aligned}$$

which implies

$$\langle \tilde{x},\tilde{y}\rangle \leq 1$$

or, removing the normalization,

$$\mathrm{re}\left\{\left\langle \underline{x},\underline{y}\right\rangle\right\} \leq \|x\|\cdot\|y\|$$

The same derivation holds if $\underline{x}$ is replaced by $-\underline{x}$ yielding

$$-\mathrm{re}\left\{\left\langle \underline{x},\underline{y}\right\rangle\right\} \leq \|\underline{x}\|\cdot\|\underline{y}\|$$

The last two equations imply

$$\left|\left\langle \underline{x},\underline{y}\right\rangle\right| \leq \|\underline{x}\|\cdot\|\underline{y}\|$$

The complex case can be shown by rotating the components of $\underline{x}$ and $\underline{y}$ such that $\mathrm{re}\left\{\langle \tilde{x},\tilde{y}\rangle\right\}$ becomes equal to $|\langle \tilde{x},\tilde{y}\rangle|$.

6.6.4 Triangle Inequality

The *triangle inequality* states that the length of any side of a triangle is less than or equal to the sum of the lengths of the other two sides, with equality occurring only when the triangle degenerates to a line. In $\mathbf{C}^N$, this becomes

$$\|\underline{x}+\underline{y}\| \leq \|\underline{x}\| + \|\underline{y}\|$$

We can show this quickly using the Schwarz Inequality.

$$\begin{aligned}
\|\underline{x}+\underline{y}\|^2 &= \left\langle \underline{x}+\underline{y},\underline{x}+\underline{y}\right\rangle \\
&= \|\underline{x}\|^2 + 2\mathrm{re}\left\{\left\langle \underline{x},\underline{y}\right\rangle\right\} + \|\underline{y}\|^2 \\
&\leq \|\underline{x}\|^2 + 2\left|\left\langle \underline{x},\underline{y}\right\rangle\right| + \|\underline{y}\|^2 \\
&\leq \|\underline{x}\|^2 + 2\|\underline{x}\|\cdot\|\underline{y}\| + \|\underline{y}\|^2 \\
&= \left(\|\underline{x}\| + \|\underline{y}\|\right)^2 \\
\Longrightarrow \quad \|\underline{x}+\underline{y}\| &\leq \|\underline{x}\| + \|\underline{y}\|
\end{aligned}$$

6.6.5 Triangle Difference Inequality

A useful variation on the triangle inequality is that the length of any side of a triangle is *greater* than the *absolute difference* of the lengths of the other two sides:

$$\|\underline{x} - \underline{y}\| \geq \left|\|\underline{x}\| - \|\underline{y}\|\right|$$

Proof:

$$\begin{aligned}
\|\underline{x} - \underline{y}\|^2 &= \|\underline{x}\|^2 - 2\mathrm{re}\left\{\langle \underline{x}, \underline{y}\rangle\right\} + \|\underline{y}\|^2 \\
&\geq \|\underline{x}\|^2 - 2\left|\langle \underline{x}, \underline{y}\rangle\right| + \|\underline{y}\|^2 \\
&\geq \|\underline{x}\|^2 - 2\|\underline{x}\| \cdot \|\underline{y}\| + \|\underline{y}\|^2 \\
&= \left[\|\underline{x}\| - \|\underline{y}\|\right]^2 \\
\Longrightarrow \quad \|\underline{x} - \underline{y}\| &\geq \left|\|\underline{x}\| - \|\underline{y}\|\right|
\end{aligned}$$

6.6.6 Vector Cosine

The Cauchy-Schwarz Inequality can be written

$$\frac{\left|\langle \underline{x}, \underline{y}\rangle\right|}{\|\underline{x}\| \cdot \|\underline{y}\|} \leq 1$$

In the case of real vectors $\underline{x}, \underline{y}$, we can always find a real number θ which satisfies

$$\cos(\theta) \stackrel{\Delta}{=} \frac{\langle \underline{x}, \underline{y}\rangle}{\|\underline{x}\| \cdot \|\underline{y}\|}$$

We thus interpret θ as the *angle* between two vectors in $\mathbf{R}^N$. In $\mathbf{C}^N$ we can similarly define $|\cos(\theta)|$.

6.6.7 Orthogonality

The vectors (signals) x and y are said to be *orthogonal* if $\langle x, y\rangle = 0$, denoted $x \perp y$. That is to say

$$\boxed{x \perp y \Leftrightarrow \langle x, y\rangle = 0.}$$

Note that if x and y are real and orthogonal, the cosine of the angle between them is zero. In plane geometry ($N = 2$), the angle between two perpendicular lines is $\pi/2$, and $\cos(\pi/2) = 0$, as expected. More

generally, orthogonality corresponds to the fact that two vectors in N-space intersect at a *right angle* and are thus *perpendicular* geometrically.

Example ($N = 2$):

Let $x = [1, 1]$ and $y = [1, -1]$, as shown in Fig. 6.8.

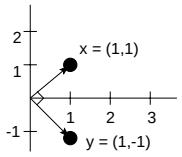

Figure 6.8: Example of two orthogonal vectors for $N = 2$.

The inner product is $\langle x, y\rangle = 1 \cdot \overline{1} + 1 \cdot \overline{(-1)} = 0$. This shows that the vectors are *orthogonal*. As marked in the figure, the lines intersect at a right angle and are therefore perpendicular.

6.6.8 The Pythagorean Theorem in N-Space

In 2D, the Pythagorean Theorem says that when x and y are orthogonal, as in Fig. 6.8, (i.e., when the triangle formed by x, y, and $x + y$, with y translated to the tip of x, is a *right triangle*), then we have

$$\|x + y\|^2 = \|x\|^2 + \|y\|^2.$$

This relationship generalizes to N dimensions, as we can easily show:

$$\begin{aligned}
\|x + y\|^2 &= \langle x + y, x + y\rangle \\
&= \langle x, x\rangle + \langle x, y\rangle + \langle y, x\rangle + \langle y, y\rangle \\
&= \|x\|^2 + \langle x, y\rangle + \overline{\langle x, y\rangle} + \|y\|^2 \\
&= \|x\|^2 + \|y\|^2 + 2\mathrm{re}\left\{\langle x, y\rangle\right\}
\end{aligned}$$

If $x \perp y$, then $\langle x, y\rangle = 0$ and the Pythagorean Theorem $\|x + y\|^2 = \|x\|^2 + \|y\|^2$ holds in N dimensions. If, on the other hand, we assume the Pythagorean Theorem holds, then since all norms are positive unless x or y is zero, we must have $\langle x, y\rangle = 0$. Finally, if x or y is zero, the result holds trivially.

Note that we also have an alternate version of the Pythagorean theorem:

$$\|x - y\|^2 = \|x\|^2 + \|y\|^2 \iff x \perp y.$$

6.6.9 Projection

The *orthogonal projection* (or simply "projection") of $\underline{y} \in \mathbf{C}^N$ onto $\underline{x} \in \mathbf{C}^N$ is defined by

$$\mathcal{P}_{\underline{x}}(\underline{y}) \stackrel{\Delta}{=} \frac{\langle \underline{y}, \underline{x} \rangle}{\|\underline{x}\|^2} \underline{x}$$

The complex scalar $\langle \underline{x}, \underline{y} \rangle / \|\underline{x}\|^2$ is called the *coefficient of projection*. When projecting $\underline{y}$ onto a *unit length* vector $\underline{x}$, the coefficient of projection is simply the inner product of $\underline{y}$ with $\underline{x}$.

Motivation: The basic idea of orthogonal projection of $\underline{y}$ onto $\underline{x}$ is to "drop a perpendicular" from $\underline{y}$ onto $\underline{x}$ to define a new vector along $\underline{x}$ which we call the "projection" of $\underline{y}$ onto $\underline{x}$. This is illustrated for $N = 2$ in Fig. 6.9 for $\underline{x} = [4, 1]$ and $\underline{y} = [2, 3]$, in which case

$$\mathcal{P}_{\underline{x}}(\underline{y}) \stackrel{\Delta}{=} \frac{\langle \underline{y}, \underline{x} \rangle}{\|\underline{x}\|^2} \underline{x} = \frac{(2 \cdot \overline{4} + 3 \cdot \overline{1})}{4^2 + 1^2} \underline{x} = \frac{11}{17} \underline{x} = \left[\frac{44}{17}, \frac{11}{17} \right]$$

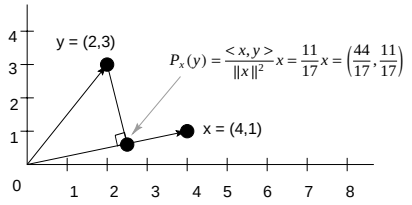

Figure 6.9: Projection of $\underline{y}$ onto $\underline{x}$ in 2D space.

Derivation: (1) Since any projection onto $\underline{x}$ must lie along the line colinear with $\underline{x}$, write the projection as $\mathcal{P}_{\underline{x}}(\underline{y}) = \alpha \underline{x}$. (2) Since by definition the projection is orthogonal to $\underline{x}$, we must have

$$\begin{aligned} (\underline{y} - \alpha \underline{x}) &\perp \underline{x} \\ \Leftrightarrow \langle \underline{y} - \alpha \underline{x}, \underline{x} \rangle &= 0 \\ \Leftrightarrow \langle \underline{y}, \underline{x} \rangle &= \alpha \langle \underline{x}, \underline{x} \rangle \\ \Leftrightarrow \alpha &= \frac{\langle \underline{y}, \underline{x} \rangle}{\langle \underline{x}, \underline{x} \rangle} = \frac{\langle \underline{y}, \underline{x} \rangle}{\|\underline{x}\|^2} \end{aligned}$$

6.7 Signal Reconstruction from Projections

We now know how to project a signal onto other signals. We now need to learn how to reconstruct a signal $\underline{x} \in \mathbf{C}^N$ from its projections onto N

different vectors $\underline{s}_k$, $k = 0, 1, 2, \ldots, N-1$. This will give us the *inverse DFT* operation (or the inverse of whatever transform we are working with).

As a simple example, consider the projection of a signal $x \in \mathbf{C}^N$ onto the rectilinear *coordinate axes* of $\mathbf{C}^N$. The coordinates of the projection onto the 0th coordinate axis are simply $(x_0, 0, \ldots, 0)$. The projection along coordinate axis 1 has coordinates $(0, x_1, 0, \ldots, 0)$, and so on. The original signal x is then clearly the *vector sum* of its projections onto the coordinate axes:

$$x = (x_0, \ldots, x_{N-1}) = (x_0, 0, \ldots, 0) + (0, x_1, 0, \ldots, 0) + \cdots (0, \ldots, 0, x_{N-1})$$

To make sure the previous paragraph is understood, let's look at the details for the case $N = 2$. We want to project an arbitrary two-sample signal $\underline{x} = (x_0, x_1)$ onto the coordinate axes in 2D. A coordinate axis can be represented by any nonzero vector along its length. The horizontal axis can be represented by any vector of the form $(\alpha, 0)$ while the vertical axis can be represented by any vector of the form $(0, \beta)$. For maximum simplicity, let's choose the positive unit-length representatives:

$$\begin{aligned} \underline{e}_0 &\stackrel{\Delta}{=} (1, 0) \\ \underline{e}_1 &\stackrel{\Delta}{=} (0, 1) \end{aligned}$$

The projection of $\underline{x}$ onto $\underline{e}_0$ is by definition

$$\mathcal{P}_{\underline{e}_0}(\underline{x}) \stackrel{\Delta}{=} \frac{\langle \underline{x}, \underline{e}_0 \rangle}{\|\underline{e}_0\|^2} \underline{e}_0 = \langle \underline{x}, \underline{e}_0 \rangle \, \underline{e}_0 = \langle [x_0, x_1], [1, 0] \rangle \, \underline{e}_0 = (x_0 \cdot \overline{1} + x_1 \cdot \overline{0}) \underline{e}_0 = x_0 \underline{e}_0 = [x_0, 0]$$

Similarly, the projection of $\underline{x}$ onto $\underline{e}_1$ is

$$\mathcal{P}_{\underline{e}_1}(\underline{x}) \stackrel{\Delta}{=} \frac{\langle \underline{x}, \underline{e}_1 \rangle}{\|\underline{e}_1\|^2} \underline{e}_1 = \langle \underline{x}, \underline{e}_1 \rangle \, \underline{e}_1 = \langle [x_0, x_1], [0, 1] \rangle \, \underline{e}_1 = (x_0 \cdot \overline{0} + x_1 \cdot \overline{1}) \underline{e}_1 = x_1 \underline{e}_1 = [0, x_1]$$

The *reconstruction* of x from its projections onto the coordinate axes is then the *vector sum of the projections*:

$$\underline{x} = \mathcal{P}_{\underline{e}_0}(\underline{x}) + \mathcal{P}_{\underline{e}_1}(\underline{x}) = x_0 \underline{e}_0 + x_1 \underline{e}_1 \stackrel{\Delta}{=} x_0(1, 0) + x_1(0, 1) = (x_0, x_1)$$

The projection of a vector onto its coordinate axes is in some sense trivial because the very meaning of the *coordinates* is that they are scalars

x_n to be applied to the *coordinate vectors* $\underline{e}_n$ in order to form an arbitrary vector $\underline{x} \in \mathbf{C}^N$ as a *linear combination* of the coordinate vectors:

$$\underline{x} \stackrel{\Delta}{=} x_0\underline{e}_0 + x_1\underline{e}_1 + \cdots + x_{N-1}\underline{e}_{N-1}$$

Note that the coordinate vectors are *orthogonal*. Since they are also unit length, $\|\underline{e}_n\| = 1$, we say that the coordinate vectors $\{\underline{e}_n\}_{n=0}^{N-1}$ are *orthonormal*.

What's more interesting is when we project a signal $\underline{x}$ onto a set of vectors *other than* the coordinate set. This can be viewed as a *change of coordinates* in $\mathbf{C}^N$. In the case of the DFT, the new vectors will be chosen to be *sampled complex sinusoids*.

6.7.1 An Example of Changing Coordinates in 2D

As a simple example, let's pick the following pair of new coordinate vectors in 2D

$$\begin{aligned}\underline{s}_0 &\stackrel{\Delta}{=} [1,1]\\ \underline{s}_1 &\stackrel{\Delta}{=} [1,-1]\end{aligned}$$

These happen to be the DFT sinusoids for $N = 2$ having frequencies $f_0 = 0$ ("dc") and $f_1 = f_s/2$ (half the sampling rate). (The sampled complex sinusoids of the DFT reduce to real numbers only for $N = 1$ and $N = 2$.) We already showed in an earlier example that these vectors are *orthogonal*. However, they are not orthonormal since the norm is $\sqrt{2}$ in each case. Let's try projecting $\underline{x}$ onto these vectors and seeing if we can reconstruct by summing the projections.

The projection of $\underline{x}$ onto $\underline{s}_0$ is by definition

$$\mathcal{P}_{\underline{s}_0}(\underline{x}) \stackrel{\Delta}{=} \frac{\langle \underline{x}, \underline{s}_0\rangle}{\|\underline{s}_0\|^2}\underline{s}_0 = \frac{\langle [x_0,x_1],[1,1]\rangle}{2}\underline{s}_0 = \frac{(x_0\cdot\overline{1} + x_1\cdot\overline{1})}{2}\underline{s}_0 = \frac{x_0+x_1}{2}\underline{s}_0$$

Similarly, the projection of $\underline{x}$ onto $\underline{s}_1$ is

$$\mathcal{P}_{\underline{s}_1}(\underline{x}) \stackrel{\Delta}{=} \frac{\langle \underline{x}, \underline{s}_1\rangle}{\|\underline{s}_1\|^2}\underline{s}_1 = \frac{\langle [x_0,x_1],[1,-1]\rangle}{2}\underline{s}_1 = \frac{(x_0\cdot\overline{1} - x_1\cdot\overline{1})}{2}\underline{s}_1 = \frac{x_0-x_1}{2}\underline{s}_1$$

The sum of these projections is then

$$\begin{aligned}
\mathcal{P}_{\underline{s}_0}(\underline{x}) + \mathcal{P}_{\underline{s}_1}(\underline{x}) &= \frac{x_0 + x_1}{2}\underline{s}_0 + \frac{x_0 - x_1}{2}\underline{s}_1 \\
&\triangleq \frac{x_0 + x_1}{2}(1,1) + \frac{x_0 - x_1}{2}(1,-1) \\
&= \left(\frac{x_0 + x_1}{2} + \frac{x_0 - x_1}{2}, \frac{x_0 + x_1}{2} - \frac{x_0 - x_1}{2}\right) \\
&= (x_0, x_1) \triangleq \underline{x}
\end{aligned}$$

It worked!

Now consider another example:

$$\begin{aligned}
\underline{s}_0 &\triangleq [1,1] \\
\underline{s}_1 &\triangleq [-1,-1]
\end{aligned}$$

The projections of $x = [x_0, x_1]$ onto these vectors are

$$\begin{aligned}
\mathcal{P}_{\underline{s}_0}(\underline{x}) &= \frac{x_0 + x_1}{2}\underline{s}_0 \\
\mathcal{P}_{\underline{s}_1}(\underline{x}) &= -\frac{x_0 + x_1}{2}\underline{s}_1
\end{aligned}$$

The sum of the projections is

$$\begin{aligned}
\mathcal{P}_{\underline{s}_0}(\underline{x}) + \mathcal{P}_{\underline{s}_1}(\underline{x}) &= \frac{x_0 + x_1}{2}\underline{s}_0 - \frac{x_0 + x_1}{2}\underline{s}_1 \\
&\triangleq \frac{x_0 + x_1}{2}(1,1) - \frac{x_0 + x_1}{2}(-1,-1) \\
&= \left(\frac{x_0 + x_1}{2}, \frac{x_0 + x_1}{2}\right) \neq \underline{x}
\end{aligned}$$

Something went wrong, but what? It turns out that a set of N vectors can be used to reconstruct an arbitrary vector in $\mathbf{C}^N$ from its projections only if they are *linearly independent*. In general, a set of vectors is *linearly independent* if none of them can be expressed as a linear combination of the others in the set. What this means intuituvely is that they must "point in different directions" in N space. In this example $s_1 = -s_0$ so that they lie along the *same line* in N-space. As a result, they are linearly *dependent*: one is a linear combination of the other.

Consider this example:

$$\begin{aligned}
\underline{s}_0 &\triangleq [1,1] \\
\underline{s}_1 &\triangleq [0,1]
\end{aligned}$$

These point in different directions, but they are not orthogonal. What happens now? The projections are

$$\begin{aligned}\mathcal{P}_{\underline{s}_0}(\underline{x}) &= \frac{x_0 + x_1}{2}\underline{s}_0 \\ \mathcal{P}_{\underline{s}_1}(\underline{x}) &= x_1\underline{s}_0\end{aligned}$$

The sum of the projections is

$$\begin{aligned}\mathcal{P}_{\underline{s}_0}(\underline{x}) + \mathcal{P}_{\underline{s}_1}(\underline{x}) &= \frac{x_0 + x_1}{2}\underline{s}_0 + x_1\underline{s}_1 \\ &\triangleq \frac{x_0 + x_1}{2}(1,1) + x_1(0,1) \\ &= \left(\frac{x_0 + x_1}{2}, \frac{x_0 + 3x_1}{2}\right) \\ &\neq \underline{x}\end{aligned}$$

So, even though the vectors are linearly independent, the sum of projections onto them does not reconstruct the original vector. Since the sum of projections worked in the orthogonal case, and since orthogonality implies linear independence, we might conjecture at this point that the sum of projections onto a set of N vectors will reconstruct the original vector only when the vector set is *orthogonal*, and this is true, as we will show.

It turns out that one can apply an orthogonalizing process, called *Gram-Schmidt orthogonalization* to any N linearly independent vectors in $\mathbf{C}^N$ so as to form an orthogonal set which will always work. This will be derived in Section 6.7.3.

Obviously, there must be at least N vectors in the set. Otherwise, there would be too few *degrees of freedom* to represent an arbitrary $\underline{x} \in \mathbf{C}^N$. That is, given the N coordinates $\{x(n)\}_{n=0}^{N-1}$ of $\underline{x}$ (which are scale factors relative to the coordinate vectors $\underline{e}_n$ in $\mathbf{C}^N$), we have to find at least N coefficients of projection (which we may think of as coordinates relative to new coordinate vectors $\underline{s}_k$). If we compute only $M < N$ coefficients, then we would be mapping a set of N complex numbers to $M < N$ numbers. Such a mapping cannot be invertible in general. It also turns out N linearly independent vectors is always sufficient. The next section will summarize the general results along these lines.

6.7.2 General Conditions

This section summarizes and extends the above derivations in a somewhat formal manner (following portions of chapter 4 of [16]).

Definition: A set of vectors is said to form a *vector space* if given any two members $\underline{x}$ and $\underline{y}$ from the set, the vectors $\underline{x}+\underline{y}$ and $c\underline{x}$ are also in the set, where c is any scalar.

Vectors defined as a list of N complex numbers $\underline{x} = (x_0, \ldots, x_{N-1}) \in \mathbf{C}^N$, using elementwise addition and multiplication by complex scalars $c \in \mathcal{C}$, form a vector space, where $\mathbf{C}^N$ denotes the set of all length N complex sequences. Similarly, real N-vectors $\underline{x} \in \mathbf{R}^N$ and real scalars $\alpha \in \mathbf{R}$ form a vector space.

Theorem: The set of all linear combinations of any set of vectors from $\mathbf{R}^N$ or $\mathbf{C}^N$ forms a vector space.

Proof: Let the original set of vectors be denoted $\underline{s}_0, \ldots, \underline{s}_{M-1}$, where M can be any integer greater than zero. Then any member $\underline{x}_0$ of the vector space is by definition a linear combination of them:

$$\underline{x}_0 = \alpha_0\underline{s}_0 + \cdots + \alpha_0\underline{s}_{M-1}$$

From this we see that $c\underline{x}_0$ is also a linear combination of the original vectors, and hence is in the vector space. Also, given any second vector from the space $\underline{x}_1 = \alpha_1\underline{s}_0 + \cdots + \alpha_1\underline{s}_{M-1}$, the sum is

$$\begin{aligned}\underline{x}_0 + \underline{x}_1 &= (\alpha_0\underline{s}_0 + \cdots \alpha_0\underline{s}_{M-1}) + (\alpha_1\underline{s}_0 + \cdots \alpha_1\underline{s}_{M-1}) \\ &= (\alpha_0+\alpha_1)\underline{s}_0 + \cdots + (\alpha_0+\alpha_1)\underline{s}_{M-1}\end{aligned}$$

which is just another linear combination of the original vectors, and hence is in the vector space. It is clear here how the closure of vector addition and scalar multiplication are "inherited" from the closure of real or complex numbers under addition and multiplication.

Definition: If a vector space consists of the set of all linear combinations of a finite set of vectors $\underline{s}_0, \ldots, \underline{s}_{M-1}$, then those vectors are said to *span* the space.

Example: The *coordinate vectors* in $\mathbf{C}^N$ span $\mathbf{C}^N$ since every vector $\underline{x} \in \mathbf{C}^N$ can be expressed as a linear combination of the coordinate vectors as

$$\underline{x} = x_0\underline{e}_0 + x_1\underline{e}_1 + \cdots + x_{N-1}\underline{e}_{N-1}$$

where $x_i \in \mathcal{C}$, and $\underline{e}_0 = (1,0,0,\ldots,0)$, $\underline{e}_1 = (0,1,0,\ldots,0)$, $\underline{e}_2 = (0,0,1,0,\ldots,0)$, and so on up to $\underline{e}_{N-1} = (0,\ldots,0,1)$.

Theorem: (i) If $\underline{s}_0, \ldots, \underline{s}_{M-1}$ span a vector space, and if one of them, say $\underline{s}_m$, is linearly dependent on the others, then the vector space is

spanned by the set obtained by omitting $\underline{s}_m$ from the original set. (ii) If $\underline{s}_0, \ldots, \underline{s}_{M-1}$ span a vector space, we can always select from these a linearly independent set that spans the same space.

Proof: Any $\underline{x}$ in the space can be represented as a linear combination of the vectors $\underline{s}_0, \ldots, \underline{s}_{M-1}$. By expressing $\underline{s}_m$ as a linear combination of the other vectors in the set, the linear combination for $\underline{x}$ becomes a linear combination of vectors other than $\underline{s}_m$. Thus, $\underline{s}_m$ can be eliminated from the set, proving (i). To prove (ii), we can define a procedure for forming the require subset of the original vectors: First, assign $\underline{s}_0$ to the set. Next, check to see if $\underline{s}_0$ and $\underline{s}_1$ are linearly dependent. If so (i.e., $\underline{s}_1$ is a scalar times $\underline{s}_0$), then discard $\underline{s}_1$; otherwise assign it also to the new set. Next, check to see if $\underline{s}_2$ is linearly dependent on the vectors in the new set. If it is (i.e., $\underline{s}_1$ is some linear combination of $\underline{s}_0$ and $\underline{s}_1$) then discard it; otherwise assign it also to the new set. When this procedure terminates after processing $\underline{s}_{M-1}$, the new set will contain only linearly independent vectors which span the original space.

Definition: A set of linearly independent vectors which spans a vector space is called a *basis* for that vector space.

Definition: The set of coordinate vectors in $\mathbf{C}^N$ is called the *natural basis* for $\mathbf{C}^N$, where the nth basis vector is

$$\underline{e}_n = (0, \ldots, 0, \underbrace{1}_{n\text{th}}, 0, \ldots, 0).$$

Theorem: The linear combination expressing a vector in terms of basis vectors for a vector space is *unique*.

Proof: Suppose a vector $\underline{x} \in \mathbf{C}^N$ can be expressed in two different ways as a linear combination of basis vectors $\underline{s}_0, \ldots, \underline{s}_{N-1}$:

$$\begin{aligned} \underline{x} &= \alpha_0 \underline{s}_0 + \cdots \alpha_{N-1} \underline{s}_{N-1} \\ &= \beta_0 \underline{s}_0 + \cdots \beta_{N-1} \underline{s}_{N-1} \end{aligned}$$

where $\alpha_i \neq \beta_i$ for at least one value of $i \in [0, N-1]$. Subtracting the two representations gives

$$\underline{0} = (\alpha_0 - \beta_0)\underline{s}_0 + \cdots + (\alpha_{N-1} - \beta_{N-1})\underline{s}_{N-1}$$

Since the vectors are linearly independent, it is not possible to cancel the nonzero vector $(\alpha_i - \beta_i)\underline{s}_i$ using some linear combination of the other vectors in the sum. Hence, $\alpha_i = \beta_i$ for all $i = 0, 1, 2, \ldots, N-1$.

Note that while the linear combination relative to a particular basis is unique, the choice of basis vectors is not. For example, given any basis set in $\mathbf{C}^N$, a new basis can be formed by *rotating* all vectors in $\mathbf{C}^N$ by the same angle. In this way, an infinite number of basis sets can be generated.

As we will soon show, the DFT can be viewed as a *change of coordinates* from coordinates relative to the *natural basis* in $\mathbf{C}^N$, $\{\underline{e}_n\}_{n=0}^{N-1}$, to coordinates relative to the *sinusoidal basis* for $\mathbf{C}^N$, $\{\underline{s}_k\}_{k=0}^{N-1}$, where $\underline{s}_k(n) \triangleq e^{j\omega_k t_n}$. The sinusoidal basis set for $\mathbf{C}^N$ consists of length N sampled complex sinusoids at frequencies $\omega_k = 2\pi k f_s/N, k = 0, 1, 2, \ldots, N-1$. Any scaling of these vectors in $\mathbf{C}^N$ by complex scale factors could also be chosen as the sinusoidal basis (i.e., any nonzero amplitude and any phase will do). However, for simplicity, we will only use unit-amplitude, zero-phase complex sinusoids as the Fourier "frequency-domain" basis set. To summarize this paragraph, the time-domain samples of a signal are its coordinates relative to the natural basis for $\mathbf{C}^N$, while its spectral coefficients are the coordinates of the signal relative to the sinusoidal basis for $\mathbf{C}^N$.

Theorem: Any two bases of a vector space contain the same number of vectors.

Proof: Left as an exercise (or see [16]).

Definition: The number of vectors in a basis for a particular space is called the *dimension* of the space. If the dimension is N, the space is said to be an N *dimensional space*, or N*-space*.

In this reader, we will only consider finite-dimensional vector spaces in any detail. However, the discrete-time Fourier transform (DTFT) and Fourier transform both require infinite-dimensional basis sets, because there is an infinite number of points in both the time and frequency domains.

Theorem: The projections of any vector $\underline{x} \in \mathbf{C}^N$ onto any orthogonal basis set for $\mathbf{C}^N$ can be summed to reconstruct $\underline{x}$ exactly.

Proof: Let $\{\underline{s}_0, \ldots, \underline{s}_{N-1}\}$ denote any orthogonal basis set for $\mathbf{C}^N$. Then since $\underline{x}$ is in the space spanned by these vectors, we have

$$\underline{x} = \alpha_0\underline{s}_0 + \alpha_1\underline{s}_1 + \cdots + \alpha_{N-1}\underline{s}_{N-1}$$

for some (unique) scalars $\alpha_0, \ldots, \alpha_{N-1}$. The projection of $\underline{x}$ onto $\underline{s}_k$ is equal to

$$\mathcal{P}_{\underline{s}_k}(\underline{x}) = \alpha_0\mathcal{P}_{\underline{s}_k}(\underline{s}_0) + \alpha_1\mathcal{P}_{\underline{s}_k}(\underline{s}_1) + \cdots + \alpha_{N-1}\mathcal{P}_{\underline{s}_k}(\underline{s}_{N-1})$$

(using the linearity of the projection operator which follows from linearity of the inner product in its first argument). Since the basis vectors are orthogonal, the projection of $\underline{s}_l$ onto $\underline{s}_k$ is zero for $l \neq k$:

$$\mathcal{P}_{\underline{s}_k}(\underline{s}_l) \triangleq \frac{\langle \underline{s}_l, \underline{s}_k \rangle}{\| \underline{s}_k \|^2} \underline{s}_k = \begin{cases} \underline{0}, & l \neq k \\ \underline{s}_k, & l = k \end{cases}$$

We therefore obtain

$$\mathcal{P}_{\underline{s}_k}(\underline{x}) = 0 + \cdots + 0 + \alpha_k \mathcal{P}_{\underline{s}_k}(\underline{s}_k) + 0 + \cdots + 0 = \alpha_k \underline{s}_k$$

Therefore, the sum of projections onto the vectors $\underline{s}_k$ is just the linear combination of the $\underline{s}_k$ which forms $\underline{x}$.

6.7.3 Gram-Schmidt Orthogonalization

Theorem: Given a set of N linearly independent vectors $\underline{s}_0, \ldots, \underline{s}_{N-1}$ from $\mathbf{C}^N$, we can construct an *orthonormal* set $\underline{\tilde{s}}_0, \ldots, \underline{\tilde{s}}_{N-1}$ which are linear combinations of the original set and which span the same space.

Proof: We prove the theorem by constructing the desired orthonormal set $\{\underline{\tilde{s}}_k\}$ sequentially from the original set $\{\underline{s}_k\}$. This procedure is known as *Gram-Schmidt orthogonalization*.

1. Set $\underline{\tilde{s}}_0 \triangleq \frac{\underline{s}_0}{\| \underline{s}_0 \|}$.
2. Define $\underline{y}_1$ as the $\underline{s}_1$ minus the projection of $\underline{s}_1$ onto $\underline{\tilde{s}}_0$:

   $$\underline{y}_1 \triangleq \underline{s}_1 - \mathcal{P}_{\underline{\tilde{s}}_0}(\underline{s}_1) = \underline{s}_1 - \langle \underline{s}_1, \underline{\tilde{s}}_0 \rangle \, \underline{\tilde{s}}_0$$

   The vector $\underline{y}_1$ is orthogonal to $\underline{\tilde{s}}_0$ by construction. (We subtracted out the part of $\underline{s}_1$ that wasn't orthogonal to $\underline{\tilde{s}}_0$.)
3. Set $\underline{\tilde{s}}_1 \triangleq \frac{\underline{y}_1}{\| \underline{y}_1 \|}$ (i.e., normalize the result of the preceding step).
4. Define $\underline{y}_2$ as the $\underline{s}_2$ minus the projection of $\underline{s}_2$ onto $\underline{\tilde{s}}_0$ and $\underline{\tilde{s}}_1$:

   $$\underline{y}_2 \triangleq \underline{s}_2 - \mathcal{P}_{\underline{\tilde{s}}_0}(\underline{s}_2) - \mathcal{P}_{\underline{\tilde{s}}_1}(\underline{s}_2) = \underline{s}_2 - \langle \underline{s}_2, \underline{\tilde{s}}_0 \rangle \, \underline{\tilde{s}}_0 - \langle \underline{s}_2, \underline{\tilde{s}}_1 \rangle \, \underline{\tilde{s}}_1$$

5. Normalize: $\underline{\tilde{s}}_2 \triangleq \frac{\underline{y}_2}{\| \underline{y}_2 \|}$.
6. Continue this process until $\underline{\tilde{s}}_{N-1}$ has been defined.

The Gram-Schmidt orthogonalization procedure will construct an orthonormal basis from any set of N linearly independent vectors. Obviously, by skipping the normalization step, we could also form simply an orthogonal basis. The key ingredient of this procedure is that each new orthonormal basis vector is obtained by subtracting out the projection of the next linearly independent vector onto the vectors accepted so far in the set. We may say that each new linearly independent vector $\underline{s}_k$ is projected onto the *subspace* spanned by the vectors $\{\tilde{\underline{s}}_0, \ldots, \tilde{\underline{s}}_{k-1}\}$, and any nonzero projection in that subspace is subtracted out of $\underline{s}_k$ to make it orthogonal to the entire subspace. In other words, we retain only that portion of each new vector $\underline{s}_k$ which points along a new dimension. The first direction is arbitrary and is determined by whatever vector we choose first ($\underline{s}_0$ here). The next vector is forced to be orthogonal to the first. The second is forced to be orthogonal to the first two, and so on.

This chapter can be considered an introduction to some of the most important concepts from linear algebra. The student is invited to pursue further reading in any textbook on linear algebra, such as [16].

6.8 Appendix: Matlab Examples

Here's how Fig. 6.1 was generated in Matlab:

```
>> x = [2 3];                  % coordinates of x
>> origin = [0 0];             % coordinates of the origin
>> xcoords = [origin(1) x(1)]; % plot() expects coordinate lists
>> ycoords = [origin(2) x(2)];
>> plot(xcoords,ycoords);      % Draw a line from origin to x
```

Mathematica can plot a list of ordered pairs:

```
In[1]:
    (* Draw a line from (0,0) to (2,3): *)
    ListPlot[{{0,0},{2,3}},PlotJoined->True];
```

In Matlab, the *mean* of the row-vector x can be computed as

```
x * ones(size(x'))/N
```

or by using the built-in function `mean()`.

In Matlab, if `x = [x1 ... xN]` is a row vector, we can compute the *total energy* as

$$\mathcal{E}_x = \texttt{x*x'}.$$

Matlab has a function `orth()` which will compute an orthonormal basis for a space given any set of vectors which span the space.

```
>> help orth

 ORTH   Orthogonalization.
        Q = orth(A) is an orthonormal basis for the range of A.
        Q'*Q = I, the columns of Q span the same space as the columns
        of A and the number of columns of Q is the rank of A.

        See also QR, NULL.
```

Below is an example of using `orth()` to orthonormalize a linearly independent basis set for $N = 3$:

```
% Demonstration of the Matlab function orth() for
% taking a set of vectors and returning an orthonormal set
% which span the same space.
v1 = [1; 2; 3];  % our first basis vector (a column vector)
v2 = [1; -2; 3]; % a second, linearly independent column vector
v1' * v2         % show that v1 is not orthogonal to v2

ans =

     6

V = [v1,v2]      % Each column of V is one of our vectors

V =

     1     1
     2    -2
     3     3

W = orth(V)      % Find an orthonormal basis for the same space

W =

    0.2673    0.1690
    0.5345   -0.8452
```

```
    0.8018    0.5071

w1 = W(:,1)      % Break out the returned vectors

w1 =

    0.2673
    0.5345
    0.8018

w2 = W(:,2)

w2 =

    0.1690
   -0.8452
    0.5071

w1' * w2  % Check that w1 is orthogonal to w2 (to working precision)

ans =

   2.5723e-17

w1' * w1  % Also check that the new vectors are unit length in 3D

ans =

     1

w2' * w2

ans =

     1

W' * W    % faster way to do the above checks (matrix multiplication)

ans =
```

```
    1.0000    0.0000
    0.0000    1.0000


% Construct some vector x in the space spanned by v1 and v2:
x = 2 * v1 - 3 * v2

x =

    -1
    10
    -3

% Show that x is also some linear combination of w1 and w2:
c1 = x' * w1      % Coefficient of projection of x onto w1

c1 =

    2.6726

c2 = x' * w2      % Coefficient of projection of x onto w2

c2 =

  -10.1419

xw = c1 * w1 + c2 * w2  % Can we make x using w1 and w2?

xw =

   -1.0000
   10.0000
   -3.0000

error = x - xw

error =
```

```
   1.0e-14 *

    0.1332
         0
         0

norm(error)       % typical way to summarize a vector error

ans =

   1.3323e-15

% It works!

% Now, construct some vector x NOT in the space spanned by v1 and v2:
y = [1; 0; 0];     % Almost anything we guess in 3D will work

%  Try to express y as a linear combination of w1 and w2:
c1 = y' * w1;      % Coefficient of projection of y onto w1
c2 = y' * w2;      % Coefficient of projection of y onto w2
yw = c1 * w1 + c2 * w2  % Can we make y using w1 and w2?

yw =

    0.1000
    0.0000
    0.3000

yerror = y - yw

yerror =

    0.9000
    0.0000
   -0.3000

norm(yerror)

ans =
```

```
    0.9487

% While the error is not zero, it is the smallest possible
% error in the least squares sense.
% That is, yw is the optimal least-squares approximation
% to y in the space spanned by v1 and v2 (w1 and w2).
% In other words, norm(yerror) <= norm(y-yw2) for any other vector yw2 made
% using a linear combination of v1 and v2.
% In yet other words, we obtain the optimal least squares approximation
% of y (which lives in 3D) in some subspace W (a 2D subspace of 3D)
% by projecting y orthogonally onto the subspace W to get yw as above.
%
% An important property of the optimal least-squares approximation
% is that the approximation error is orthogonal to the the subspace
% in which the approximation lies.  Let's show this:

W' * yerror % must be zero to working precision

ans =

   1.0e-16 *

   -0.2574
   -0.0119
```

Chapter 7

Derivation of the Discrete Fourier Transform (DFT)

This chapter derives the Discrete Fourier Transform (DFT) as a projection of a length N signal $x(\cdot)$ onto the set of N sampled complex sinusoids generated by the N roots of unity.

7.1 The DFT Derived

In this section, the Discrete Fourier Transform (DFT) will be derived.

7.1.1 Geometric Series

Recall that for any compex number $z_1 \in \mathcal{C}$, the signal $x(n) \stackrel{\Delta}{=} z_1^n$, $n = 0, 1, 2, \ldots$, defines a *geometric sequence*, i.e., each term is obtained by multiplying the previous term by a (complex) constant. A *geometric series* is defined as the *sum* of a geometric sequence:

$$S_N(z_1) \stackrel{\Delta}{=} \sum_{n=0}^{N-1} z_1^n = 1 + z_1 + z_1^2 + z_1^3 + \cdots + z_1^{N-1}$$

If $z_1 \neq 1$, the sum can be expressed in *closed form* as

$$\boxed{S_N(z_1) = \frac{1 - z_1^N}{1 - z_1}}$$

Proof: We have

$$\begin{aligned} S_N(z_1) &\stackrel{\Delta}{=} 1 + z_1 + z_1^2 + z_1^3 + \cdots + z_1^{N-1} \\ z_1 S_N(z_1) &= z_1 + z_1^2 + z_1^3 + \cdots + z_1^{N-1} + z_1^N \\ \Longrightarrow \quad S_N(z_1) - z_1 S_N(z_1) &= 1 - z_1^N \\ \Longrightarrow \quad S_N(z_1) &= \frac{1 - z_1^N}{1 - z_1} \end{aligned}$$

7.1.2 Orthogonality of Sinusoids

A key property of sinusoids is that they are *orthogonal at different frequencies*. That is,

$$\omega_1 \neq \omega_2 \implies A_1 \sin(\omega_1 t + \phi_1) \perp A_2 \sin(\omega_2 t + \phi_2)$$

This is true whether they are complex or real, and whatever amplitude and phase they may have. All that matters is that the frequencies be different. Note, however, that the sinusoidal durations *must* be infinity.

For length N *sampled* sinuoidal signal segments, such as used by the DFT, exact orthogonality holds only for the *harmonics of the sampling rate divided by* N, i.e., only over the frequencies $f_k = kf_s/N, k = 0, 1, 2, 3, \ldots, N-1$. These are the only frequencies that have an *exact integer number of periods in* N *samples* (depicted in Fig. 7.2 for $N = 8$).

The complex sinusoids corresponding to the frequencies f_k are

$$s_k(n) \stackrel{\Delta}{=} e^{j\omega_k nT}, \quad \omega_k \stackrel{\Delta}{=} k\frac{2\pi}{N} f_s, \quad k = 0, 1, 2, \ldots, N-1$$

These sinusoids are generated by the N *roots of unity* in the complex plane:

$$W_N^k \stackrel{\Delta}{=} e^{j\omega_k T} \stackrel{\Delta}{=} e^{jk2\pi(f_s/N)T} = e^{jk2\pi/N}, \quad k = 0, 1, 2, \ldots, N-1$$

These are called the N roots of unity because each of them satisfies

$$\left[W_N^k\right]^N = \left[e^{j\omega_k T}\right]^N = \left[e^{jk2\pi/N}\right]^N = e^{jk2\pi} = 1$$

The N roots of unity are plotted in the complex plane in Fig. 7.1 for $N = 8$. In general, for any N, there will always be a point at $z = 1$, and the points equally subdivide the unit circle. When N is even, there is a

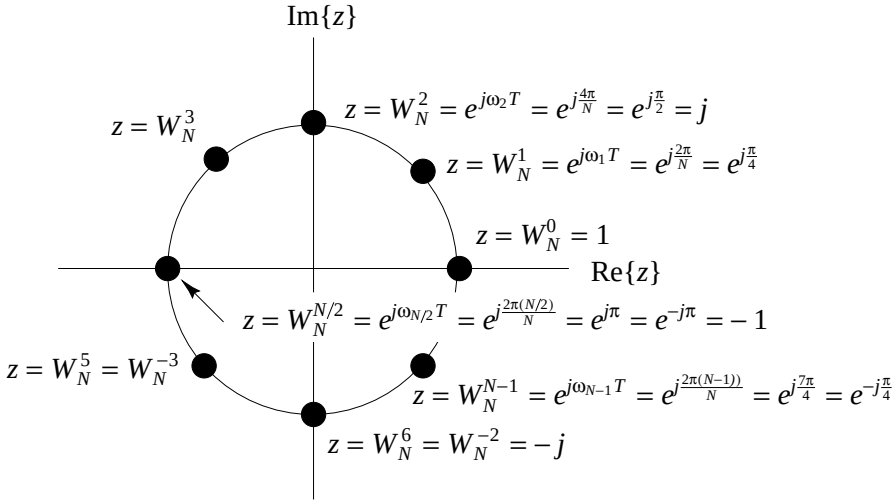

Figure 7.1: The N roots of unity for $N = 8$.

point at $z = -1$ (corresponding to a sinusoid at exactly half the sampling rate), while if N is odd, there is no point at $z = -1$.

The sampled sinusoids corresponding to the N roots of unity are plotted in Fig. 7.2. These are the sampled sinusoids $(W_N^k)^n = e^{j2\pi kn/N} = e^{j\omega_k nT}$ used by the DFT. Note that taking successively higher integer powers of the point W_N^k on the unit circle *generates* samples of the kth DFT sinusoid, giving $[W_N^k]^n$, $n = 0, 1, 2, \ldots, N-1$. The kth sinusoid generator W_N^k is in turn the kth power of the *primitive Nth root of unity* $W_N \triangleq e^{j2\pi/N}$. The notation W_N, W_N^k, and W_N^{nk} are common in the digital signal processing literature.

Note that in Fig. 7.2 the range of k is taken to be $[-N/2, N/2-1] = [-4, 3]$ instead of $[0, N-1] = [0, 7]$. This is the most "physical" choice since it corresponds with our notion of "negative frequencies." However, we may add any integer multiple of N to k without changing the sinusoid indexed by k. In other words, $k \pm mN$ refers to the same sinusoid for all integer m.

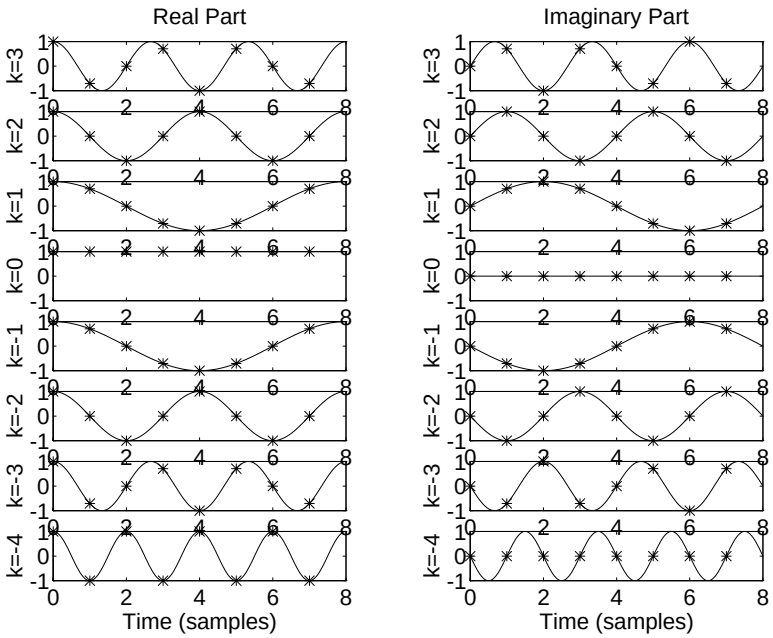

Figure 7.2: Complex sinusoids used by the DFT for $N = 8$.

7.1.3 Orthogonality of the DFT Sinusoids

We now show mathematically that the DFT sinusoids are exactly orthogonal. Let

$$s_k(n) \stackrel{\Delta}{=} e^{j\omega_k nT} = e^{j2\pi kn/N}$$

denote the kth complex DFT sinusoid. Then

$$\begin{aligned}\langle s_k, s_l\rangle &\stackrel{\Delta}{=} \sum_{n=0}^{N-1} s_k(n)\overline{s_l(n)} = \sum_{n=0}^{N-1} e^{j2\pi kn/N} e^{-j2\pi ln/N} \\ &= \sum_{n=0}^{N-1} e^{j2\pi(k-l)n/N} = \frac{1-e^{j2\pi(k-l)}}{1-e^{j2\pi(k-l)/N}}\end{aligned}$$

where the last step made use of the closed-form expression for the sum of a geometric series. If $k \neq l$, the denominator is nonzero while the numerator is zero. This proves

$$\boxed{s_k \perp s_l, \quad k \neq l}$$

While we only looked at unit amplitude, zero phase complex sinusoids, as used by the DFT, it is readily verified that the (nonzero) amplitude and phase have no effect on orthogonality.

7.1.4 Norm of the DFT Sinusoids

For $k = l$, we follow the previous derivation to the next-to-last step to get

$$\langle s_k, s_k\rangle = \sum_{n=0}^{N-1} e^{j2\pi(k-k)n/N} = N$$

which proves

$$\boxed{\| s_k \| = \sqrt{N}}$$

7.1.5 An Orthonormal Sinusoidal Set

We can normalize the DFT sinusoids to obtain an orthonormal set:

$$\underline{\tilde{s}}_k(n) \stackrel{\Delta}{=} \frac{s_k(n)}{\sqrt{N}} = \frac{e^{j2\pi kn/N}}{\sqrt{N}}$$

The orthonormal sinusoidal basis signals satisfy

$$\langle \tilde{\underline{s}}_k, \tilde{\underline{s}}_l \rangle = \begin{cases} 1, & k = l \\ 0, & k \neq l \end{cases}$$

We call these the *normalized DFT sinusoids*.

7.1.6 The Discrete Fourier Transform (DFT)

Given a signal $x(\cdot) \in \mathbf{C}^N$, the *spectrum* is defined by

$$X(\omega_k) \triangleq \langle x, s_k \rangle = \sum_{n=0}^{N-1} x(n)\overline{s_k(n)}, \quad k = 0, 1, 2, \ldots, N-1$$

or, as is most often written

$$\boxed{X(\omega_k) \triangleq \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}}, \quad k = 0, 1, 2, \ldots, N-1}$$

That is, the kth sample $X(\omega_k)$ of the spectrum of x is defined as the inner product of x with the kth DFT sinusoid s_k. This definition is N times the *coefficient of projection* of x onto s_k, i.e.,

$$\frac{\langle x, s_k \rangle}{\| s_k \|^2} = \frac{X(\omega_k)}{N}$$

The projection of x onto s_k itself is

$$\mathcal{P}_{s_k}(x) = \frac{X(\omega_k)}{N} s_k$$

The inverse DFT is simply the sum of the projections:

$$x = \sum_{k=0}^{N-1} \frac{X(\omega_k)}{N} s_k$$

or, as we normally write,

$$\boxed{x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(\omega_k) e^{j\frac{2\pi kn}{N}}}$$

In summary, the DFT is proportional to the set of coefficients of projection onto the sinusoidal basis set, and the inverse DFT is the reconstruction of the original signal as a superposition of its sinusoidal projections.

7.1.7 Frequencies in the "Cracks"

The DFT is defined only for frequencies $\omega_k = 2\pi k f_s/N$. If we are analyzing one or more periods of an exactly periodic signal, where the period is exactly N samples (or some integer divisor of N), then these really are the only frequencies present in the signal, and the spectrum is actually zero everywhere but at $\omega = \omega_k$. However, we use the DFT to analyze arbitrary signals from nature. What happens when a frequency ω is present in a signal x that is not one of the DFT-sinusoid freqencies ω_k?

To find out, let's project a length N segment of a sinusoid at an arbitrary freqency ω onto the kth DFT sinusoid:

$$\begin{aligned} x(n) &\triangleq e^{j\omega nT} \\ s_k(n) &\triangleq e^{j\omega_k nT} \\ \mathcal{P}_{s_k}(x) &= \frac{\langle x, s_k\rangle}{\langle s_k, s_k\rangle} s_k \triangleq \frac{X(\omega_k)}{N} s_k \end{aligned}$$

The coefficient of projection is proportional to

$$\begin{aligned} X(\omega_k) &\triangleq \langle x, s_k\rangle \triangleq \sum_{n=0}^{N-1} x(n)\overline{s_k(n)} \\ &= \sum_{n=0}^{N-1} e^{j\omega nT} e^{-j\omega_k nT} = \sum_{n=0}^{N-1} e^{j(\omega-\omega_k)nT} = \frac{1-e^{j(\omega-\omega_k)NT}}{1-e^{j(\omega-\omega_k)T}} \\ &= e^{j(\omega-\omega_k)(N-1)T/2} \frac{\sin[(\omega-\omega_k)NT/2]}{\sin[(\omega-\omega_k)T/2]} \end{aligned}$$

using the closed-form expression for a geometric series sum. As previously shown, the sum is N at $\omega = \omega_k$ and zero at ω_l, for $l \neq k$. However, the sum is *nonzero at all other frequencies*.

Since we are only looking at N samples, any sinusoidal segment can be projected onto the N DFT sinusoids and be reconstructed exactly by a linear combination of them. Another way to say this is that the DFT sinusoids form a *basis* for $\mathbf{C}^N$, so that any length N signal whatsoever can be expressed as linear combination of them. Therefore, when analyzing segments of recorded signals, we must interpret what we see accordingly.

The typical way to think about this in practice is to consider the DFT operation as a *digital filter*.[1] The *frequency response* of this filter is what

[1] More precisely, $\mathrm{DFT}_k()$ is a length N finite-impulse-response (FIR) digital filter. See §8.7 for related discussion.

we just computed,[2] and its magnitude is

$$|X(\omega_k)| = \left|\frac{\sin[(\omega-\omega_k)NT/2]}{\sin[(\omega-\omega_k)T/2]}\right|$$

(shown in Fig. 7.3a for $k = N/4$). At all other integer values of k, the response is the same but shifted (circularly) left or right so that the peak is centered on ω_k. The secondary peaks away from ω_k are called *sidelobes* of the DFT response, while the main peak may be called the *main lobe* of the response. Since we are normally most interested in spectra from an audio perspective, the same plot is repeated using a *decibel* vertical scale in Fig. 7.3b (clipped at -60 dB). We see that the sidelobes are really quite high from an audio perspective. Sinusoids with frequencies near $\omega_{k\pm1.5}$, for example, are only attenuated approximately 13 dB in the DFT output $X(\omega_k)$.

We see that $X(\omega_k)$ is sensitive to *all* frequencies between dc and the sampling rate *except* the other DFT-sinusoid frequencies ω_l for $l \neq k$. This is sometimes called *spectral leakage* or *cross-talk* in the spectrum analysis. Again, there is *no error* when the signal being analyzed is truly periodic and we can choose N to be exactly a period, or some multiple of a period. Normally, however, this cannot be easily arranged, and spectral leakage can really become a problem.

Note that spectral leakage is not reduced by increasing N. It can be thought of as being caused by abruptly *truncating* a sinusoid at the beginning and/or end of the N-sample time window. Only the DFT sinusoids are not cut off at the window boundaries. All other frequencies will suffer some truncation distortion, and the spectral content of the abrupt cut-off or turn-on transient can be viewed as the source of the sidelobes. Remember that, as far as the DFT is concerned, the input signal $x(n)$ is the same as its *periodic extension*. If we repeat N samples of a sinusoid at frequency $\omega \neq \omega_k$, there will be a "glitch" every N samples since the signal is not periodic in N samples. This glitch can be considered a source of new energy over the entire spectrum.

To reduce spectral leakage (cross-talk from far-away frequencies), we typically use a *window* function, such as a "raised cosine" window, to *taper* the data record gracefully to zero at both endpoints of the window. As a result of the smooth tapering, the *main lobe widens* and the *sidelobes decrease* in the DFT response. Using no window is better viewed as using

[2] We call this the *aliased sinc function* to distinguish it from the *sinc function* $\mathrm{sinc}(x) \triangleq \sin(\pi x)/(\pi x)$.

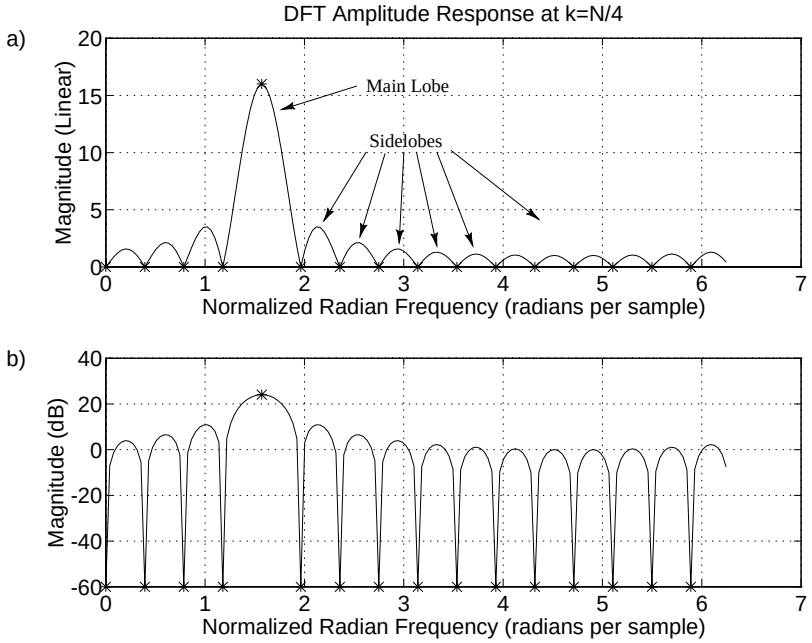

Figure 7.3: Frequency response magnitude of a single DFT output sample.

a *rectangular window* of length N, unless the signal is exactly periodic in N samples. These topics are considered further in Music 420 and in the "Examples using the DFT" chapter.

Since the kth spectral sample $X(\omega_k)$ is properly regarded as a measure of spectral amplitude over a *range* of frequencies, nominally $k-1/2$ to $k+1/2$, this range is sometimes called a *frequency bin* (as in a "storage bin" for spectral energy). The frequency index k is called the *bin number*, and $|X(\omega_k)|^2$ can be regarded as the total energy in the kth bin (see Parseval's Theorem in the "Fourier Theorems" chapter). Similar remarks apply to samples of any continuous bandlimited function; however, the term "bin" is only used in the frequency domain, even though it could be assigned exactly the same meaning mathematically in the time domain.

7.1.8 Normalized DFT

A more "theoretically clean" DFT is obtained by projecting onto the *normalized DFT sinusoids*

$$\tilde{s}_k(n) \triangleq \frac{e^{j2\pi kn/N}}{\sqrt{N}}$$

In this case, the normalized DFT of x is

$$\tilde{X}(\omega_k) \triangleq \langle x, \tilde{s}_k \rangle = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}$$

which is also precisely the coefficient of projection of x onto $\tilde{s}_k$. The inverse normalized DFT is then more simply

$$x(n) = \sum_{k=0}^{N-1} X(\omega_k)\tilde{s}_k(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(\omega_k) e^{j2\pi kn/N}$$

While this definition is much cleaner from a "geometric signal theory" point of view, it is rarely used in practice since it requires more computation than the typical definition. However, note that the only difference between the forward and inverse transforms in this case is the sign of the exponent in the kernel.

7.2 The Length 2 DFT

The length 2 DFT is particularly simple, since the basis sinusoids are real:

$$\begin{aligned} \underline{s}_0 &= (1,1) \\ \underline{s}_1 &= (1,-1) \end{aligned}$$

The DFT sinusoid $\underline{s}_0$ is a sampled constant signal, while $\underline{s}_1$ is a sampled sinusoid at half the sampling rate.

Figure 7.4 illustrates the graphical relationships for the length 2 DFT of the signal $\underline{x} = [6, 2]$.

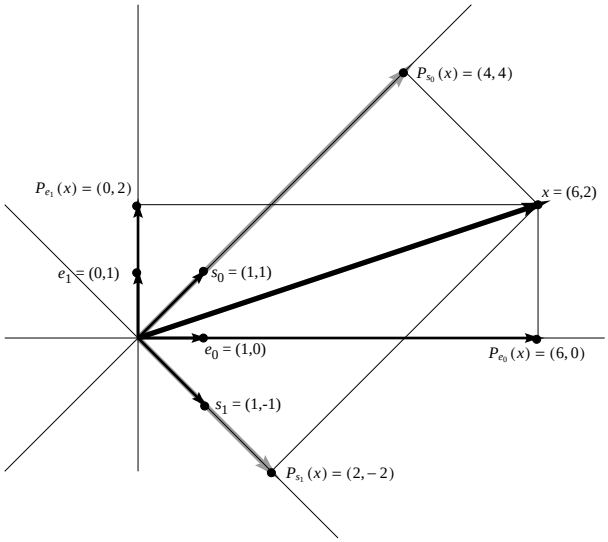

Figure 7.4: Graphical interpretation of the length 2 DFT.

Analytically, we compute the DFT to be

$$\begin{aligned} X(\omega_0) &\triangleq \mathcal{P}_{\underline{s}_0}(\underline{x}) \triangleq \frac{\langle \underline{x}, \underline{s}_0 \rangle}{\langle \underline{s}_0, \underline{s}_0 \rangle} \underline{s}_0 = \frac{6 \cdot 1 + 2 \cdot 1}{1^2 + 1^2} \underline{s}_0 = 4\underline{s}_0 = (4,4) \\ X(\omega_1) &\triangleq \mathcal{P}_{\underline{s}_1}(\underline{x}) \triangleq \frac{\langle \underline{x}, \underline{s}_1 \rangle}{\langle \underline{s}_1, \underline{s}_1 \rangle} \underline{s}_1 = \frac{6 \cdot 1 + 2 \cdot (-1)}{1^2 + (-1)^2} \underline{s}_0 = 2\underline{s}_1 = (2,-2) \end{aligned}$$

Note the lines of orthogonal projection illustrated in the figure. The "time domain" basis consists of the vectors $\{\underline{e}_0, \underline{e}_1\}$, and the orthogonal projections onto them are simply the coordinate projections $(6, 0)$

and $(0,2)$. The "frequency domain" basis vectors are $\{\underline{s}_0, \underline{s}_1\}$, and they provide an orthogonal basis set which is *rotated* 45 *degrees* relative to the time-domain basis vectors. Projecting orthogonally onto them gives $X(\omega_0) = (4,4)$ and $X(\omega_1) = (2,-2)$, respectively. The original signal $\underline{x}$ can be expressed as the vector sum of its coordinate projections (a time-domain representation), or as the vector sum of its projections onto the DFT sinusoids (a frequency-domain representation). Computing the coefficients of projection is essentially "taking the DFT" and constructing $\underline{x}$ as the vector sum of its projections onto the DFT sinusoids amounts to "taking the inverse DFT."

7.3 Matrix Formulation of the DFT

The DFT can be formulated as a complex matrix multiply, as we show in this section. For basic definitions regarding matrices, see §A.

The DFT consists of inner products of the input signal $\underline{x}$ with sampled complex sinusoidal sections $\underline{s}_k$:

$$X(\omega_k) \stackrel{\Delta}{=} \langle \underline{x}, \underline{s}_k \rangle \stackrel{\Delta}{=} \sum_{n=0}^{N-1} \underline{x}(n) e^{-j2\pi nk/N}, \quad k = 0, 1, 2, \ldots, N-1$$

By collecting the DFT output samples into a column vector, we have

$$\left[\begin{array}{c} X(\omega_0) \\ X(\omega_1) \\ X(\omega_2) \\ \vdots \\ X(\omega_{N-1}) \end{array}\right] = \left[\begin{array}{cccc} s_0(0) & s_0(1) & \cdots & s_0(N-1) \\ s_1(0) & s_1(1) & \cdots & s_1(N-1) \\ s_2(0) & s_2(1) & \cdots & s_2(N-1) \\ \vdots & \vdots & \vdots & \vdots \\ s_{N-1}(0) & s_{N-1}(1) & \cdots & s_{N-1}(N-1) \end{array}\right] \left[\begin{array}{c} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{array}\right]$$

or

$$\underline{X} = S_N \underline{x}$$

where S_N denotes the *DFT matrix* $S_N[k,n] \triangleq W_N^{-kn} \triangleq e^{-j2\pi kn/N}$, or,

$$\begin{aligned} S_N &\triangleq \begin{bmatrix} s_0(0) & s_0(1) & \cdots & s_0(N-1) \\ s_1(0) & s_1(1) & \cdots & s_1(N-1) \\ s_2(0) & s_2(1) & \cdots & s_2(N-1) \\ \vdots & \vdots & \vdots & \vdots \\ s_{N-1}(0) & s_{N-1}(1) & \cdots & s_{N-1}(N-1) \end{bmatrix} \\ &\triangleq \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{-j2\pi/N} & e^{-j4\pi/N} & \cdots & e^{-j2\pi(N-1)/N} \\ 1 & e^{-j4\pi/N} & e^{-j8\pi/N} & \cdots & e^{-j2\pi 2(N-1)/N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & e^{-j2\pi(N-1)/N} & e^{-j2\pi 2(N-1)/N} & \cdots & e^{-j2\pi(N-1)(N-1)/N} \end{bmatrix} \end{aligned}$$

We see that the kth row of the DFT matrix is the k DFT sinusoids. Since the matrix is *symmetric*, ${S_N}^{\mathrm{T}} = S_N$ (where transposition does *not* include conjugation), we observe that the kth *column* of S_N is also the kth DFT sinusoid.

The *inverse DFT matrix* is simply $\overline{S_N}/N$. That is, we can perform the inverse DFT operation simply as

$$\underline{x} = \frac{1}{N} {S_N}^* \underline{X}$$

Since $\underline{X} = S_N \underline{x}$, the above implies

$${S_N}^* S_N = N \cdot I$$

The above equation succinctly implies that the *columns of S_N are orthogonal*, which, of course, we already knew.

The *normalized DFT matrix* is given by

$$\tilde{S}_N \triangleq \frac{1}{\sqrt{N}} S_N$$

and the corresponding normalized *inverse* DFT matrix is simply $\overline{\tilde{S}_N}$, so that we have

$$\tilde{S}_N^* \tilde{S}_N = I$$

This implies that the columns of S_N are *orthonormal*. Such a matrix is said to be *unitary*.

When a *real* matrix A satisfies $A^{\mathrm{T}} A = I$, then A is said to be *orthogonal*. "Unitary" is the generalization of "orthogonal" to complex matrices.

7.4 Matlab Examples

7.4.1 Figure 7.2

Below is the Matlab for Fig. 7.2:

```
N=8;
fs=1;

n = [0:N-1]; % row
t = [0:0.01:N]; % interpolated
k=fliplr(n)' - N/2;
fk = k*fs/N;
wk = 2*pi*fk;
clf;
for i=1:N
    subplot(N,2,2*i-1);
    plot(t,cos(wk(i)*t))
    axis([0,8,-1,1]);
    hold on;
    plot(n,cos(wk(i)*n),'*')
    if i==1
title('Real Part');
    end;
    ylabel(sprintf('k=%d',k(i)));
    if i==N
xlabel('Time (samples)');
    end;
    subplot(N,2,2*i);
    plot(t,sin(wk(i)*t))
    axis([0,8,-1,1]);
    hold on;
    plot(n,sin(wk(i)*n),'*')
    ylabel(sprintf('k=%d',k(i)));
    if i==1
title('Imaginary Part');
    end;
    if i==N
xlabel('Time (samples)');
    end;
```

```
end
```

7.4.2 Figure 7.3

Below is the Matlab for Fig. 7.3:

```
% Parameters (sampling rate = 1)
N = 16;               % DFT length
k = N/4;              % bin where DFT filter is centered
wk = 2*pi*k/N;        % normalized radian center-frequency for DFT_k()
wStep = 2*pi/N;
w = [0:wStep:2*pi - wStep]; % DFT frequency grid

interp = 10;
N2 = interp*N;            % Denser grid showing "arbitrary" frequencies
w2Step = 2*pi/N2;
w2 = [0:w2Step:2*pi - w2Step]; % Extra dense frequency grid
X = (1 - exp(j*(w2-wk)*N)) ./ (1 - exp(j*(w2-wk)));
                      % slightly offset to avoid divide by zero at wk
X(1+k*interp) = N;   % Fix divide-by-zero point (overwrite "NaN")

% Plot spectral magnitude
clf;
magX = abs(X);
magXd = magX(1:interp:N2); % DFT frequencies only
subplot(2,1,1);
plot(w2,magX,'-'); hold on; grid;
plot(w,magXd,'*');         % Show DFT sample points
title('DFT Amplitude Response at k=N/4');
xlabel('Normalized Radian Frequency (radians per sample)');
ylabel('Magnitude (Linear)');
text(-1,20,'a)');

% Same thing on a dB scale
magXdb = 20*log10(magX);       % Spectral magnitude in dB
% Since the zeros go to minus infinity, clip at -60 dB:
magXdb = max(magXdb,-60*ones(1,N2));
magXddb = magXdb(1:interp:N2); % DFT frequencies only
subplot(2,1,2);
hold off; plot(w2,magXdb,'-'); hold on; plot(w,magXddb,'*'); grid;
```

```
xlabel('Normalized Radian Frequency (radians per sample)');
ylabel('Magnitude (dB)');
text(-1,40,'b)');
print -deps '../eps/dftfilter.eps';
hold off;
```

7.4.3 DFT Matrix in Matlab

The following example reinforces the discussion of the DFT matrix. We can simply create the DFT matrix in Matlab by taking the FFT of the identity matrix. Then we show that multiplying by the DFT matrix is equivalent to the FFT:

```
>> eye(4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

>> S4 = fft(eye(4))
ans =
   1.0000            1.0000            1.0000            1.0000
   1.0000            0.0000 - 1.0000i -1.0000            0.0000 + 1.0000i
   1.0000           -1.0000            1.0000           -1.0000
   1.0000            0.0000 + 1.0000i -1.0000            0.0000 - 1.0000i

>> S4' * S4          % Show that S4' = inverse DFT (times N=4)
ans =
    4.0000    0.0000         0    0.0000
    0.0000    4.0000    0.0000    0.0000
         0    0.0000    4.0000    0.0000
    0.0000    0.0000    0.0000    4.0000

>> x = [1; 2; 3; 4]
x =
     1
     2
     3
     4

>> fft(x)
ans =
  10.0000
```

```
  -2.0000 + 2.0000i
  -2.0000
  -2.0000 - 2.0000i
>> S4 * x
ans =
  10.0000
  -2.0000 + 2.0000i
  -2.0000
  -2.0000 - 2.0000i
```

Chapter 8

Fourier Theorems for the DFT

This chapter derives various *Fourier theorems* for the case of the DFT. Included are symmetry relations, the shift theorem, convolution theorem, correlation theorem, power theorem, and theorems pertaining to interpolation and downsampling. Applications related to certain theorems are outlined, including linear time-invariant filtering, sampling rate conversion, and statistical signal processing.

8.1 The DFT and its Inverse

Let $x(n), n = 0, 1, 2, \ldots, N-1$ denote an n-sample complex sequence, i.e., $x \in \mathbf{C}^N$. Then the *spectrum* of x is defined by the *Discrete Fourier Transform* (DFT):

$$\boxed{X(k) \stackrel{\Delta}{=} \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N}, \quad k = 0, 1, 2, \ldots, N-1}$$

The *inverse DFT* (*IDFT*) is defined by

$$\boxed{x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi nk/N}, \quad n = 0, 1, 2, \ldots, N-1}$$

Note that for the first time we are not carrying along the sampling interval $T = 1/f_s$ in our notation. This is actually the most typical treatment in

the digital signal processing literature. It is often said that the sampling frequency is $f_s = 1$. However, it can be set to any desired value using the substitution

$$e^{j2\pi nk/N} = e^{j2\pi k(f_s/N)nT} \stackrel{\Delta}{=} e^{j\omega_k t_n}$$

However, for the remainder of this reader, we will adopt the more common (and more mathematical) convention $f_s = 1$. In particular, we'll use the definition $\omega_k \stackrel{\Delta}{=} 2\pi k/N$ *for this chapter only*. In this case, a radian frequency ω_k is in units of "radians per sample." Elsewhere in this reader, ω_k always means "radians per *second*." (Of course, there's no difference when the sampling rate is really 1.) Another term we use in connection with the $f_s = 1$ convention is *normalized frequency*: All normalized radian frequencies lie in the range $[-\pi, \pi)$, and all normalized frequencies in Hz lie in the range $[-0.5, 0.5)$.

8.1.1 Notation and Terminology

If X is the DFT of x, we say that x and X form a *transform pair* and write

$$\boxed{x \leftrightarrow X} \quad (\text{"}x \text{ corresponds to } X\text{"}).$$

Another notation we'll use is

$$\begin{aligned} \mathrm{DFT}(x) &\stackrel{\Delta}{=} X, \quad \text{and} \\ \mathrm{DFT}_k(x) &\stackrel{\Delta}{=} X(k) \end{aligned}$$

As we've already seen, *time-domain* signals are consistently denoted using *lowercase* symbols such as "x," while *frequency-domain* signals (spectra), are denoted in *uppercase* ("X").

8.1.2 Modulo Indexing, Periodic Extension

The DFT sinusoids $s_k(n) \stackrel{\Delta}{=} e^{j\omega_k n}$ are all *periodic* having periods which divide N. That is, $s_k(n+mN) = s_k(n)$ for any integer m. Since a length N signal x can be expressed as a linear combination of the DFT sinusoids in the time domain,

$$x(n) = \frac{1}{N}\sum_k X(k)s_k(n),$$

it follows that the "automatic" definition of $x(n)$ beyond the range $[0, N-1]$ is *periodic extension*, i.e., $x(n+mN) \stackrel{\Delta}{=} x(n)$ for every integer m.

Moreover, the DFT also repeats naturally every N samples, since

$$X(k+mN) \stackrel{\Delta}{=} \sum_n \langle x, s_{k+mN}\rangle = \sum_{n=0}^{N-1} \langle x, s_k\rangle = X(k)$$

because $s_{k+mN}(n) = e^{j2\pi n(k+mN)/N} = e^{j2\pi nk/N}e^{j2\pi nm} = s_k(n)$. (The DFT sinusoids behave identically as functions of n and k.) Accordingly, for purposes of DFT studies, we may define *all* signals in $\mathbf{C}^N$ as being single periods from an infinitely long periodic signal with period N samples:

Definition: For any signal $x \in \mathbf{C}^N$, we define

$$x(n+mN) \stackrel{\Delta}{=} x(n)$$

for every integer m.

As a result of this convention, all indexing of signals and spectra[1] can be interpreted *modulo* N, and we may write $x(n \mod N)$ to emphasize this. Formally, "$n \mod N$" is defined as $n - mN$ with m chosen to give $n - mN$ in the range $[0, N-1]$.

As an example, when indexing a spectrum X, we have that $X(N) = X(0)$ which can be interpreted physically as saying that the sampling rate is the same frequency as dc for discrete time signals. In the time domain, we have what is sometimes called the "periodic extension" of $x(n)$. This means that the input to the DFT is mathematically treated as *samples of a periodic signal* with period NT seconds (N samples). The corresponding assumption in the frequency domain is that the spectrum is *zero between frequency samples* ω_k.

It is also possible to adopt the point of view that the time-domain signal $x(n)$ consists of N samples preceded and followed by *zeros*. In this case, the spectrum is *nonzero* between spectral samples ω_k, and the spectrum between samples can be reconstructed by means of *bandlimited interpolation*. This "time-limited" interpretation of the DFT input is considered in detail in Music 420 and is beyond the scope of Music 320 (except in the discussion of "zero padding $\leftrightarrow$ interpolation" below).

[1] A spectrum is mathematically identical to a signal, since both are just sequences of N complex numbers. However, for clarity, we generally use "signal" when the sequence index is considered a time index, and "spectrum" when the index is associated with successive frequency samples.

8.2 Signal Operators

It will be convenient in the Fourier theorems below to make use of the following signal operator definitions.

8.2.1 Flip Operator

Definition: We define the *flip operator* by

$$\textsc{Flip}_n(x) \stackrel{\Delta}{=} x(-n)$$

which, by modulo indexing, is $x(N-n)$. The $\textsc{Flip}()$ operator reverses the order of samples 1 through $N-1$ of a sequence, leaving sample 0 alone, as shown in Fig. 8.1a. Thanks to modulo indexing, it can also be viewed as "flipping" the sequence about the vertical axis, as shown in Fig. 8.1b. The interpretation of Fig. 8.1b is usually the one we want, and the $\textsc{Flip}$ operator is usually thought of as "time reversal" when applied to a signal x or "frequency reversal" when applied to a spectrum X.

8.2.2 Shift Operator

Definition: The *shift operator* is defined by

$$\textsc{Shift}_{\Delta,n}(x) \stackrel{\Delta}{=} x(n-\Delta)$$

and $\textsc{Shift}_\Delta(x)$ denotes the entire shifted signal. Note that since indexing is modulo N, the shift is *circular*. However, we normally use it to represent *time delay* by Δ samples. We often use the shift operator in conjunction with *zero padding* (appending zeros to the signal x) in order to avoid the "wrap-around" associated with a circular shift.

Figure 8.2 illustrates successive one-sample delays of a periodic signal having first period given by $[0,1,2,3,4]$.

Example: $\textsc{Shift}_1([1,0,0,0]) = [0,1,0,0]$ (an impulse delayed one sample).

Example: $\textsc{Shift}_1([1,2,3,4]) = [4,1,2,3]$ (a circular shift example).

Example: $\textsc{Shift}_{-2}([1,0,0,0]) = [0,0,1,0]$ (another circular shift example).

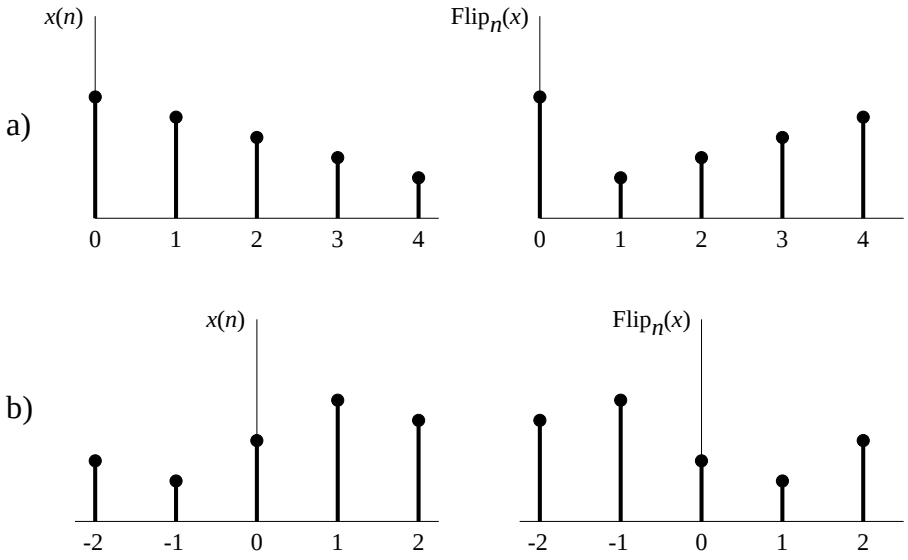

Figure 8.1: Illustration of x and $\textsc{Flip}(x)$ for $N = 5$ and two different domain interpretations:
a) $n \in [0, N-1]$.
b) $n \in [-(N-1)/2, (N-1)/2]$.

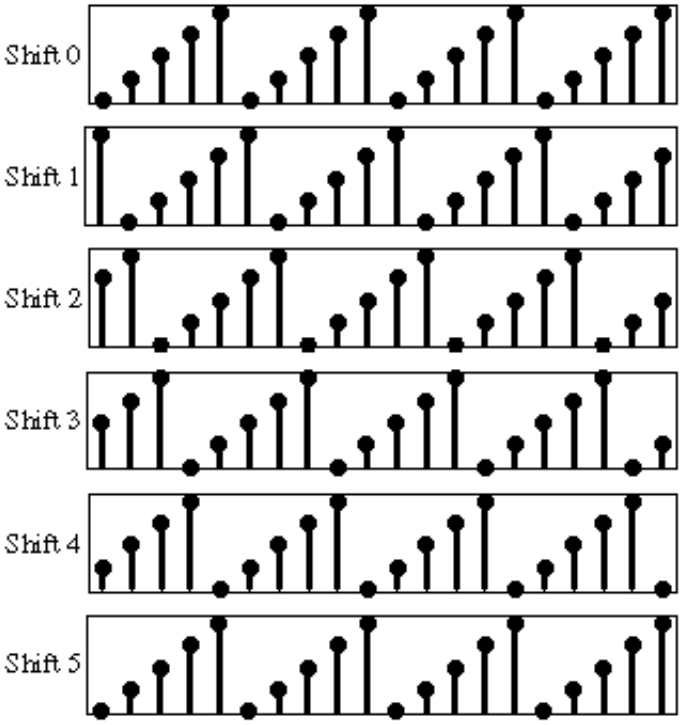

Figure 8.2: Successive one-sample shifts of a sampled periodic sawtooth waveform having first period $[0, 1, 2, 3, 4]$.

8.2.3 Convolution

Definition: The *convolution* of two signals x and y in $\mathbf{C}^N$ is denoted "$x * y$" and defined by

$$\boxed{(x * y)_n \stackrel{\Delta}{=} \sum_{m=0}^{N-1} x(m)y(n-m)}$$

Note that this is *cyclic* or "circular" convolution.[2] The importance of convolution in *linear systems theory* is discussed in §8.7

Convolution is *commutative*, i.e.,

$$\boxed{x * y = y * x}$$

Proof:

$$(x*y)_n \stackrel{\Delta}{=} \sum_{m=0}^{N-1} x(m)y(n-m) = \sum_{l=n}^{n-N+1} x(n-l)y(l) = \sum_{l=0}^{N-1} y(l)x(n-l) \stackrel{\Delta}{=} (y*x)_n$$

where in the first step we made the change of summation variable $l \stackrel{\Delta}{=} n-m$, and in the second step, we made use of the fact that any sum over all N terms is equivalent to a sum from 0 to $N-1$.

Graphical Convolution

Note that the cyclic convolution operation can be expressed in terms of previously defined operators as

$$y(n) \stackrel{\Delta}{=} (x * h)_n \stackrel{\Delta}{=} \sum_{m=0}^{N-1} x(m)h(n-m) = \langle x, \textsc{Shift}_n(\textsc{Flip}(h))\rangle \quad (h \text{ real})$$

where $x, y \in \mathbf{C}^N$ and $h \in \mathbf{R}^N$. It is instructive to interpret the last expression above *graphically*.

Figure 8.3 illustrates convolution of

$$\begin{aligned} y &= [1,1,1,1,0,0,0,0] \\ h &= [1,0,0,0,0,1,1,1] \end{aligned}$$

[2] To simulate acyclic convolution, as is appropriate for the simulation of sampled continuous-time systems, sufficient zero padding is used so that nonzero samples do not "wrap around" as a result of the shifting of y in the definition of convolution. Zero padding is discussed later in this reader.

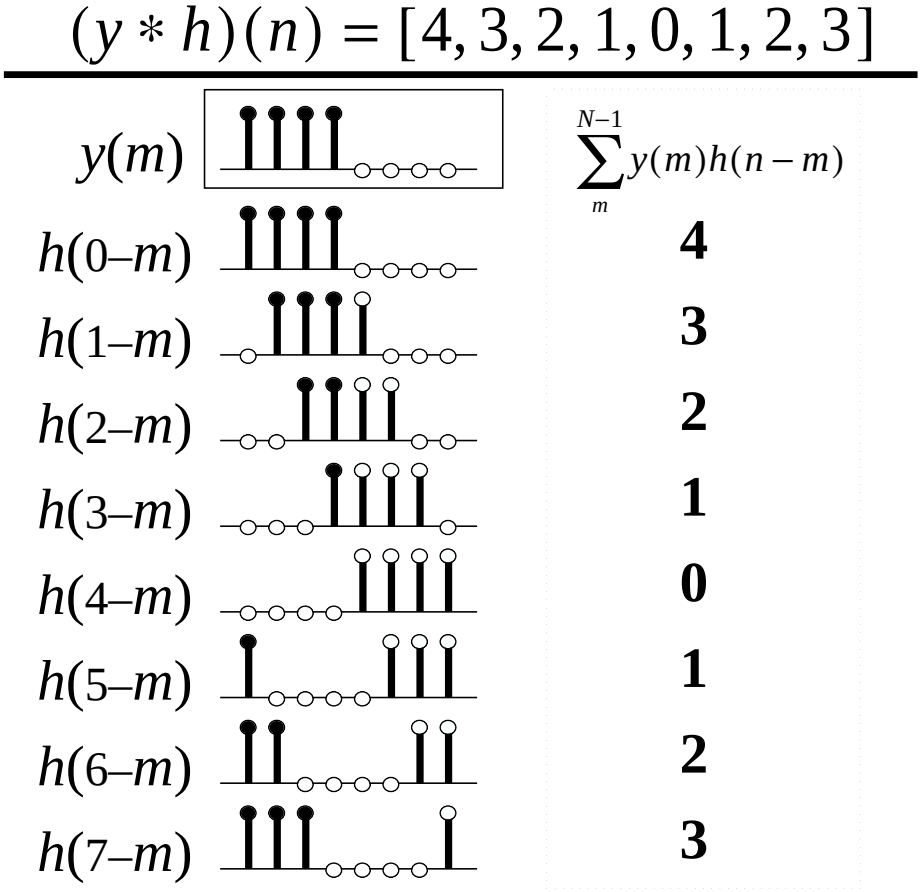

Figure 8.3: Illustration of convolution of $y = [1, 1, 1, 1, 0, 0, 0, 0]$ and its "matched filter" h=[1,0,0,0,0,1,1,1] ($N = 8$).

For example, y could be a "rectangularly windowed signal, zero-padded by a factor of 2," where the signal happened to be dc (all 1s). For the convolution, we need

$$\textsc{Flip}(h) = [1,1,1,1,0,0,0,0]$$

which is the same as y. When $h = \textsc{Flip}(y)$, we say that h is *matched filter* for y.[3] In this case, h is matched to look for a "dc component," and also zero-padded by a factor of 2. The zero-padding serves to simulate acyclic convolution using circular convolution. The figure illustrates the computation of the convolution of y and h:

$$y * h \stackrel{\Delta}{=} \sum_{n=0}^{N-1} y(n) \cdot \textsc{Flip}(h) = [4,3,2,1,0,1,2,3]$$

Note that a large peak is obtained in the convolution output at time 0. This large peak (the largest possible if all signals are limited to $[-1,1]$ in magnitude), indicates the matched filter has "found" the dc signal starting at time 0. This peak would persist even if various sinusoids at other frequencies and/or noise were added in.

Polynomial Multiplication

Note that when you multiply two polynomials together, their coefficients are *convolved*. To see this, let $p(x)$ denote the mth-order polynomial

$$p(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_m x^m$$

with coefficients p_i, and let $q(x)$ denote the nth-order polynomial

$$q(x) = q_0 + q_1 x + q_2 x^2 + \cdots + q_n x^n$$

with coefficients q_i. Then we have [17]

$$\begin{aligned} p(x)q(x) &= p_0q_0 + (p_0q_1 + p_1q_0)x + (p_0q_2 + p_1q_1 + p_2q_0)x^2 \\ &\quad +(p_0q_3 + p_1q_2 + p_2q_1 + p_3q_0)x^3 \\ &\quad +(p_0q_4 + p_1q_3 + p_2q_2 + p_3q_1 + p_4q_0)x^4 + \cdots \\ &\quad +(p_0q_{n+m} + p_1q_{n+m-1} + p_2q_{n+m-2} + p_{n+m-1}q_1 + p_{n+m}q_0)x^{n+m} \end{aligned}$$

[3] Matched filtering is briefly discussed in §8.8.

Denoting $p(x)q(x)$ by

$$r(x) \stackrel{\Delta}{=} p(x)q(x) = r_0 + r_1 x + r_2 x^2 + \cdots + r_{m+n} x^{m+n},$$

we see that the ith coefficient can be expressed as

$$\begin{aligned} r_i &= p_0 q_i + p_1 q_{i-1} + p_2 q_{i-2} + \cdots + p_{i-1} q_1 + p_i q_0 \\ &= \sum_{j=0}^{i} p_j q_{i-j} = \sum_{j=-\infty}^{\infty} p_j q_{i-j} \\ &\stackrel{\Delta}{=} (p * q)(i) \end{aligned}$$

where p_i and q_i are doubly infinite sequences, defined as zero for $i < 0$ and $i > m, n$, respectively.

Multiplication of Decimal Numbers

Since decimal numbers are implicitly just polynomials in the powers of 10, e.g.,

$$3819 = 3 \cdot 10^3 + 8 \cdot 10^2 + 1 \cdot 10^1 + 9 \cdot 10^0$$

it follows that *multiplying two numbers convolves their digits*. The only twist is that, unlike normal polynomial multiplication, we have *carries*. That is, when a convolution result exceeds 10, we subtract 10 from the result and add 1 to the digit in the next higher place.

8.2.4 Correlation

Definition: The *correlation operator* for two signals x and y in $\mathbf{C}^N$ is defined as

$$\boxed{(x \star y)_n \stackrel{\Delta}{=} \sum_{m=0}^{N-1} \overline{x(m)} y(m+n)}$$

We may interpret the correlation operator as

$$(x \star y)_n = \langle \textsc{Shift}_{-n}(y), x \rangle$$

which is the coefficient of projection onto x of y *advanced* by n samples (shifted circularly to the *left* by n samples). The time shift n is called the correlation *lag*, and $\overline{x(m)}y(m+n)$ is called a *lagged product*. Applications of correlation are discussed in §8.8.

8.2.5 Stretch Operator

Unlike all previous operators, the $\textsc{Stretch}_L()$ operator maps a length N signal to a length $M \triangleq LN$ signal, where L and N are integers. We use "m" instead of "n" as the time index to underscore this fact.

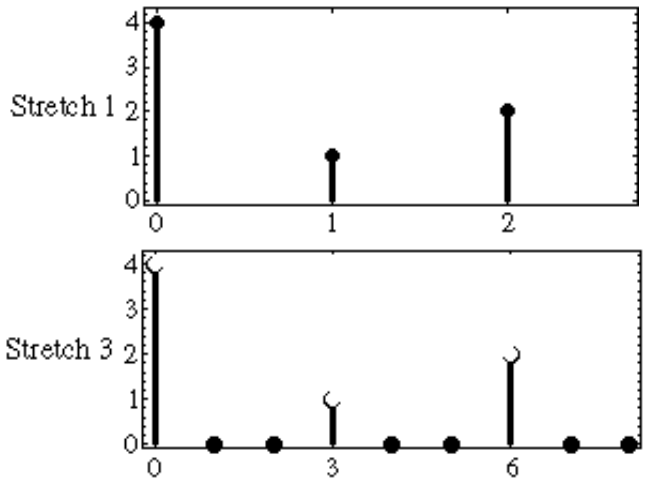

Figure 8.4: Illustration of $\textsc{Stretch}_3(x)$.

Definition: A *stretch by factor* L is defined by

$$\textsc{Stretch}_{L,m}(x) \triangleq \begin{cases} x(m/L), & m/L = \text{integer} \\ 0, & m/L \neq \text{integer} \end{cases}$$

Thus, to stretch a signal by the factor L, insert $L-1$ zeros between each pair of samples. An example of a stretch by factor three is shown in Fig. 8.4. The example is

$$\textsc{Stretch}_3([4,1,2]) = [4,0,0,1,0,0,2,0,0]$$

The stretch operator is used to describe and analyze *upsampling*, i.e., increasing the sampling rate by an integer factor.

8.2.6 Zero Padding

Definition: *Zero padding* consists of appending zeros to a signal. It maps a length N signal to a length $M > N$ signal, but M need not be

an integer multiple of N:

$$\textsc{ZeroPad}_{M,m}(x) \stackrel{\Delta}{=} \begin{cases} x(m), & 0 \le m \le N-1 \\ 0, & N \le m \le M-1 \end{cases}$$

For example,

$$\textsc{ZeroPad}_1 0([1,2,3,4,5]) = [1,2,3,4,5,0,0,0,0,0]$$

The above definition is natural when $x(n)$ represents a signal starting at time 0 and extending for N samples. If, on the other hand, we are zero-padding a spectrum, or we have a time-domain signal which has nonzero samples for negative time indices, then the zero padding is normally inserted between samples $(N-1)/2$ and $(N+1)/2$ for N odd (note that $(N+1)/2 = (N+1)/2 - N = -(N-1)/2 \mod N$), and similarly for even N. I.e., for spectra, zero padding is inserted at the point $z = -1$ ($\omega = \pi f_s$). Figure 8.5 illustrates this second form of zero padding. It is also used in conjunction with zero-phase FFT windows (discussed a bit further below).

Using Fourier theorems, we will be able to show that *zero padding in the time domain* gives *bandlimited interpolation in the frequency domain*. Similarly, *zero padding in the frequency domain* gives *bandlimited interpolation in the time domain*. This is how ideal sampling rate conversion is accomplished.

It is important to note that bandlimited interpolation is *ideal* interpolation in digital signal processing.

8.2.7 Repeat Operator

Like the $\textsc{Stretch}_L()$ operator, the $\textsc{Repeat}_L()$ operator maps a length N signal to a length $M \stackrel{\Delta}{=} LN$ signal:

Definition: The *repeat L times* operator is defined by

$$\textsc{Repeat}_{L,m}(x) \stackrel{\Delta}{=} x(m), \qquad m = 0,1,2,\ldots,M-1$$

where $M \stackrel{\Delta}{=} LN$. Thus, the $\textsc{Repeat}_L()$ operator simply repeats its input signal L times.[4] An example of $\textsc{Repeat}_2(x)$ is shown in Fig. 8.6. The

[4] You might wonder why we need this since all indexing is defined modulo N already. The answer is that $\textsc{Repeat}_L()$ formally expresses a mapping from the space of length N signals to the space of length $M = LN$ signals.

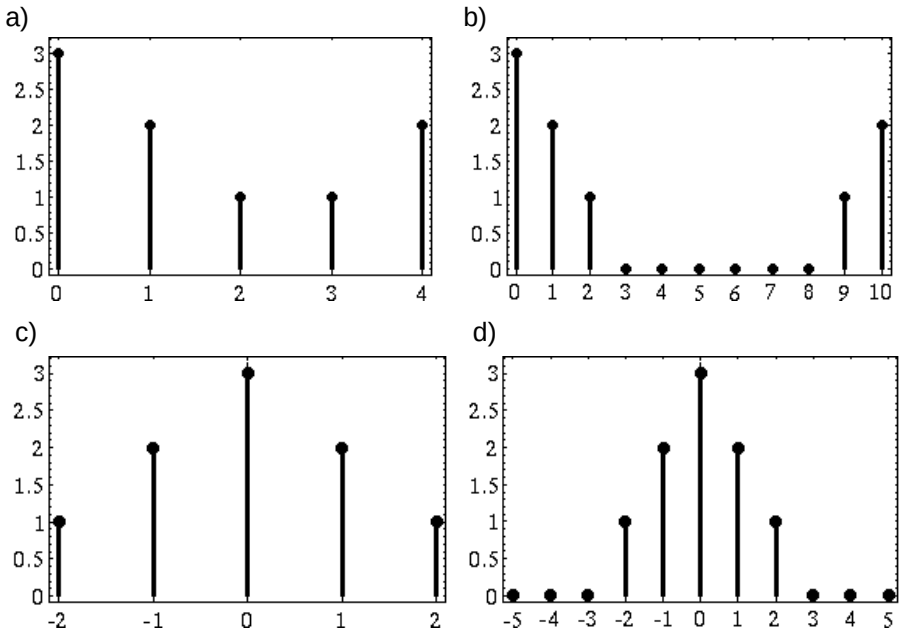

Figure 8.5: Illustration of frequency-domain zero padding:
a) Original spectrum $X = [3, 2, 1, 1, 2]$ plotted over the domain $k \in [0, N-1]$ where $N = 5$ (i.e., as the spectral array would normally exist in a computer array).
b) $\textsc{ZeroPad}_{11}(X)$.
c) The same signal X plotted over the domain $k \in [-(N-1)/2, (N-1)/2]$ which is more natural for interpreting negative frequencies.
d) $\textsc{ZeroPad}_{11}(X)$.

example is

$$\textsc{Repeat}_2([0,2,1,4,3,1]) = [0,2,1,4,3,1,0,2,1,4,3,1]$$

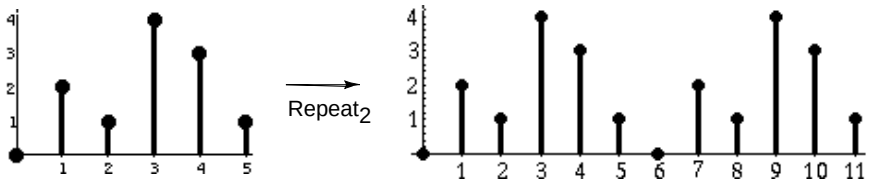

Figure 8.6: Illustration of $\textsc{Repeat}_2(x)$.

A frequency-domain example is shown in Fig. 8.7. Figure 8.7a shows the original spectrum X, Fig. 8.7b shows the same spectrum plotted over the unit circle in the z plane, and Fig. 8.7c shows $\textsc{Repeat}_3(X)$. The $z = 1$ point (dc) is on the right-rear face of the enclosing box. Note that when viewed as centered about $k = 0$, X is a somewhat "triangularly shaped" spectrum. The repeating block can be considered to extend from the point at $z = 1$ to the point far to the left, or it can be considered the triangularly shaped "baseband" spectrum centered about $z = 1$.

The repeat operator is used to state the Fourier theorem

$$\textsc{Stretch}_L \leftrightarrow \textsc{Repeat}_L$$

That is, when you stretch a signal by the factor L, its spectrum is repeated L times around the unit circle.

8.2.8 Downsampling Operator

Definition: *Downsampling by L* is defined as taking every Lth sample, starting with sample 0:

$$\textsc{Select}_{L,m}(x) \stackrel{\Delta}{=} x(mL),\ m = 0,1,2,\ldots,M-1 \quad (N = LM, x \in \mathbf{C}^N)$$

The $\textsc{Select}_L()$ operator maps a length $N = LM$ signal down to a length M signal. It is the inverse of the $\textsc{Stretch}_L()$ operator (but not vice versa), i.e.,

$$\begin{aligned} \textsc{Select}_L(\textsc{Stretch}_L(x)) &= x \\ \textsc{Stretch}_L(\textsc{Select}_L(x)) &\neq x \quad \text{(in general)}. \end{aligned}$$

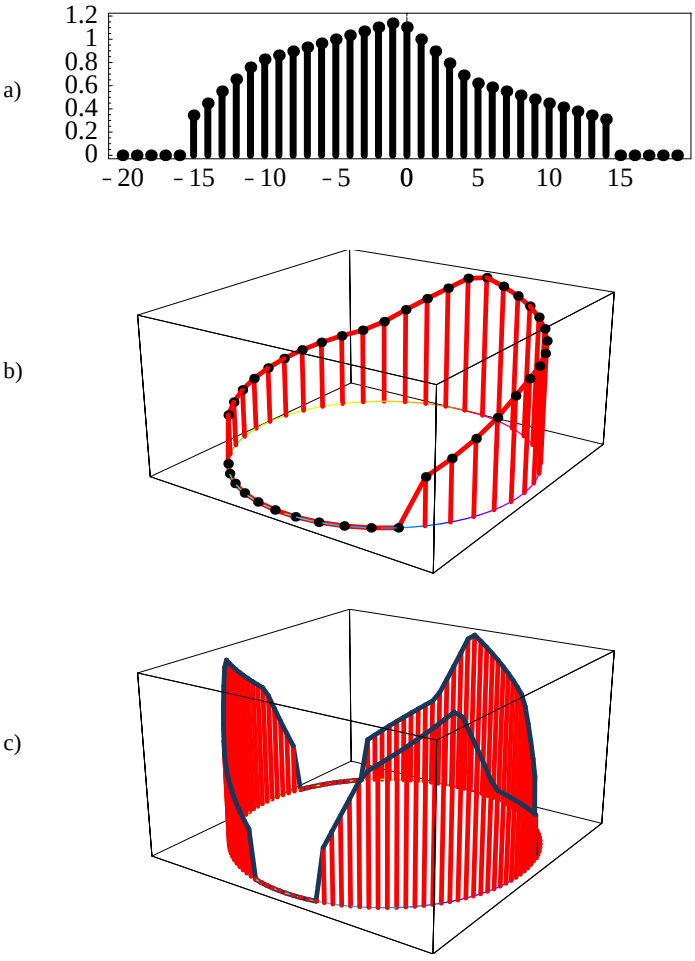

Figure 8.7: Illustration of $\textsc{Repeat}_3(X)$.
a) Conventional plot of X.
b) Plot of X over the unit circle in the z plane.
c) $\textsc{Repeat}_3(X)$.

The stretch and downsampling operations do not commute because they are linear *time-varying* operators. They can be modeled using time-varying *switches* controlled by the sample index n.

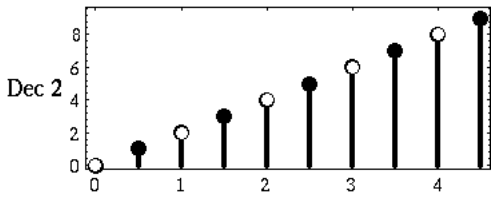

Figure 8.8: Illustration of $\textsc{Select}_2(x)$. The white-filled circles indicate the retained samples while the black-filled circles indicate the discarded samples.

An example of $\textsc{Select}_2(x)$ is shown in Fig. 8.8. The example is

$$\textsc{Select}_2([0,1,2,3,4,5,6,7,8,9]) = [0,2,4,6,8]$$

8.2.9 Alias Operator

Aliasing occurs when a signal is *undersampled*. If the signal sampling rate f_s is too low, we get *frequency-domain aliasing*.

The topic of aliasing normally arises in the context of *sampling a continuous-time signal*. Shannon's Sampling Theorem says that we will have no aliasing due to sampling as long as the sampling rate is higher than twice the highest frequency present in the signal being sampled.

In this chapter, we are considering only discrete-time signals, in order to keep the math as simple as possible. Aliasing in this context occurs when a discrete-time signal is decimated to reduce its sampling rate. You can think of continuous-time sampling as the limiting case for which the starting sampling rate is infinity.

An example of aliasing is shown in Fig. 8.9.

In the figure, the high-frequency sinusoid is indistinguishable from the lower frequency sinusoid due to aliasing. We say the higher frequency *aliases* to the lower frequency. Undersampling in the frequency domain gives rise to *time-domain aliasing*. If time or frequency is not specified, the term "aliasing" normally means frequency-domain aliasing.

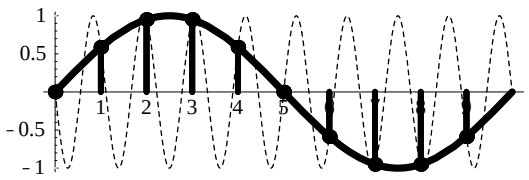

Figure 8.9: Example of aliasing due to undersampling in time.

The *aliasing operator* is defined by

$$\textsc{Alias}_{L,m}(x) \stackrel{\Delta}{=} \sum_{l=0}^{N-1} x\left(m + l\frac{N}{L}\right),\ m = 0,1,2,\ldots,M-1 \quad (N = LM, x \in \mathbf{C}^N)$$

Like the $\textsc{Select}_L()$ operator, the $\textsc{Alias}_L()$ operator maps a length $N = LM$ signal down to a length M signal. The way to think of it is to partition the original N samples in to L blocks of length M, with the first block extending from sample 0 to sample $M-1$, the second block from M to $2M-1$, etc. Then just add up the blocks. This process is called *aliasing*. If the original signal x is a time signal, it is called *time-domain aliasing*; if it is a spectrum, we call it *frequency-domain aliasing*, or just aliasing. Note that aliasing is *not invertible*. Once the blocks are added together, it is not possible to recover the original blocks, in general.

For example,

$$\begin{aligned}
\textsc{Alias}_2([0,1,2,3,4,5]) &= [0,1,2] + [3,4,5] = [3,5,7] \\
\textsc{Alias}_3([0,1,2,3,4,5]) &= [0,1] + [2,3] + [4,5] = [6,9]
\end{aligned}$$

The alias operator is used to state the Fourier theorem

$$\textsc{Select}_L \leftrightarrow \textsc{Alias}_L$$

That is, when you decimate a signal by the factor L, its spectrum is aliased by the factor L.

Figure 8.10 shows the result of $\textsc{Alias}_2$ applied to $\textsc{Repeat}_3(X)$ from Fig. 8.7c. Imagine the spectrum of Fig. 8.10a as being plotted on a piece of paper rolled to form a cylinder, with the edges of the paper meeting at $z = 1$ (upper right corner of Fig. 8.10a). Then the $\textsc{Alias}_2$ operation can be simulated by rerolling the cylinder of paper to cut its circumference in half. That is, reroll it so that at every point, *two* sheets of paper are in contact at all points on the new, narrower cylinder. Now, simply add the

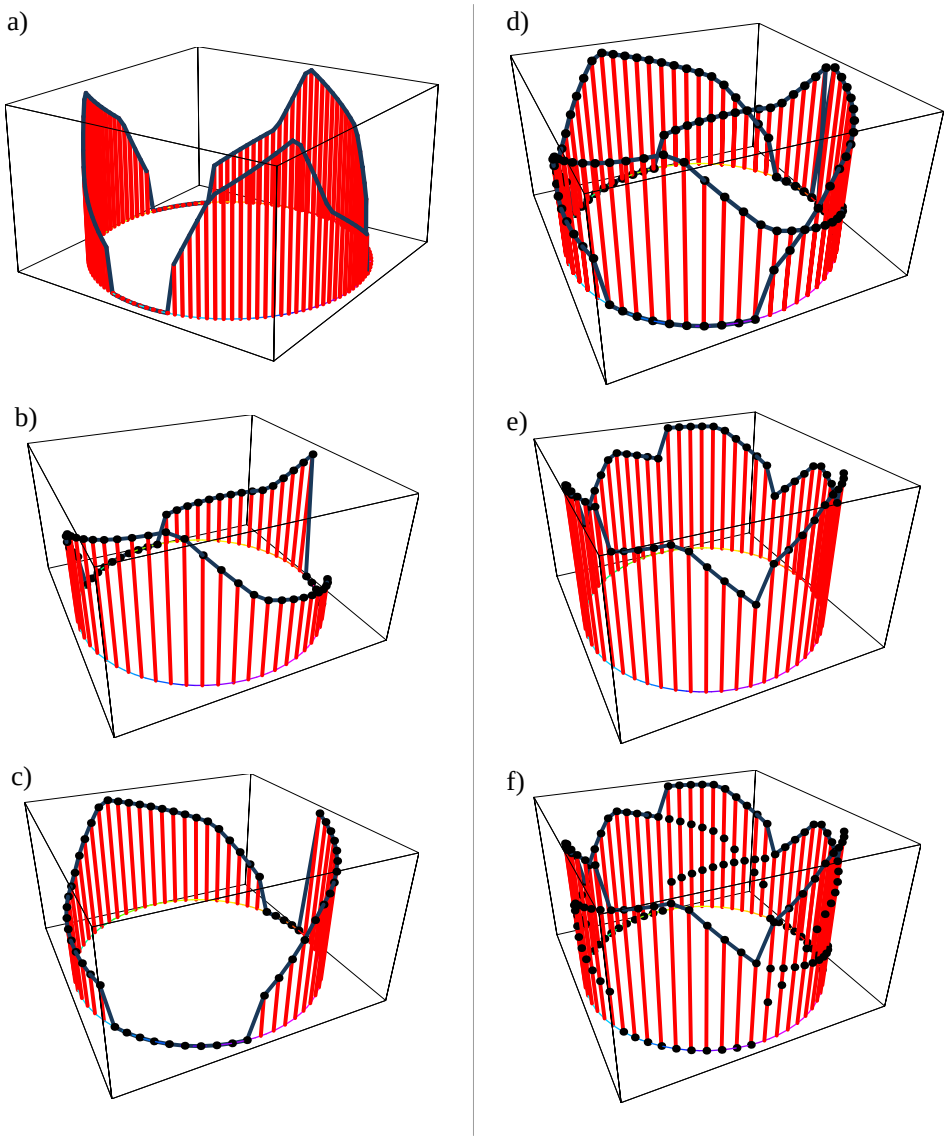

Figure 8.10: Illustration of aliasing in the frequency domain.
a) $\textsc{Repeat}_3(X)$ from Fig. 8.7c.
b) First half of the original unit circle (0 to π) wrapped around the new, smaller unit circle (which is magnified to the original size).
c) Second half (π to 2π), also wrapped around the new unit circle.
d) Overlay of components to be summed.
e) Sum of components (the aliased spectrum).
f) Both sum and overlay.

values on the two overlapping sheets together, and you have the $\textsc{Alias}_2$ of the original spectrum on the unit circle. To alias by 3, we would shrink the cylinder further until the paper edges again line up, giving three layers of paper in the cylinder, and so on.

Figure 8.10b shows what is plotted on the first circular wrap of the cylinder of paper, and Fig. 8.10c shows what is on the second wrap. These are overlaid in Fig. 8.10d and added together in Fig. 8.10e. Finally, Fig. 8.10f shows both the addition and the overlay of the two components. We say that the second component (Fig. 8.10c) "aliases" to new frequency components, while the first component (Fig. 8.10b) is considered to be at its original frequencies. If the unit circle of Fig. 8.10a covers frequencies 0 to f_s, all other unit circles (Fig. 8.10b-c) cover frequencies 0 to $f_s/2$.

In general, aliasing by the factor K corresponds to a *sampling-rate reduction* by the factor K. To prevent aliasing when reducing the sampling rate, an *anti-aliasing lowpass filter* is generally used. The lowpass filter attenuates all signal components at frequencies outside the interval $[-f_s/(2K), f_s/(2K)]$. If they are not filtered out, they will alias, and aliasing is generally non-invertible.

Conceptually, the unit circle is reduced by $\textsc{Alias}_2$ to a unit circle *half* the original size, where the two halves are summed. The inverse of aliasing is then "repeating" which should be understood as *increasing* the unit circle circumference using "periodic extension" to generate "more spectrum" for the larger unit circle. In the time domain, downsampling is the inverse of the stretch operator. All of these relationships are precise only for *integer* stretch/downsampling/aliasing/repeat factors; in continuous time, the restriction to integer factors is removed, and we obtain the (simpler) *similarity theorem* (proved in §8.9).

8.3 Even and Odd Functions

Some of the Fourier theorems can be succinctly expressed in terms of even and odd symmetries.

Definition: A function $f(n)$ is said to be *even* if $f(-n) = f(n)$. An even function is also *symmetric*, but the term symmetric applies also to functions symmetric about a point other than 0.

Definition: A function $f(n)$ is said to be *odd* if $f(-n) = -f(n)$. An odd function is also called *antisymmetric*.

Note that every odd function $f(n)$ must satisfy $f(0) = 0$. Moreover,

for any $x \in \mathbf{C}^N$ with N even, we also have $x(N/2) = 0$ since $x(N/2) = -x(-N/2) = -x(-N/2+N) = -x(N/2)$, i.e., $N/2$ and $-N/2$ index the same point.

Theorem: Every function $f(n)$ can be decomposed into a sum of its even part $f_e(n)$ and odd part $f_o(n)$, where

$$\begin{aligned} f_e(n) &\stackrel{\Delta}{=} \frac{f(n)+f(-n)}{2} \\ f_o(n) &\stackrel{\Delta}{=} \frac{f(n)-f(-n)}{2} \end{aligned}$$

Proof: In the above definitions, f_e is even and f_o is odd by construction. Summing, we have

$$f_e(n) + f_o(n) = \frac{f(n)+f(-n)}{2} + \frac{f(n)-f(-n)}{2} = f(n)$$

Theorem: The product of even functions is even, the product of odd functions is even, and the product of an even times an odd function is odd.

Proof: Readily shown.

Since even times even is even, odd times odd is even, and even times odd is odd, we can think of even as $(+)$ and odd as $(-)$: $(+)\cdot(+) = (+)$, $(-)\cdot(-) = (+)$, and $(+)\cdot(-) = (-)\cdot(+) = (-)$.

Example: $\cos(\omega_k n)$ is an *even* signal since $\cos(-\theta) = \cos(\theta)$.

Example: $\sin(\omega_k n)$ is an *odd* signal since $\sin(-\theta) = -\sin(\theta)$.

Example: $\cos(\omega_k n)\cdot\sin(\omega_l n)$ is *odd* (even times odd).

Example: $\sin(\omega_k n)\cdot\sin(\omega_l n)$ is *even* (odd times odd).

Theorem: The sum of all the samples of an odd signal x_o in $\mathbf{C}^N$ is zero.

Proof: This is readily shown by writing the sum as $x_o(0) + [x_o(1) + x_o(-1)] + \cdots + x(N/2)$, where the last term only occurs when N is even. Each term so written is zero for an odd signal x_o.

Example: For all DFT sinusoidal frequencies $\omega_k = 2\pi k/N$,

$$\sum_{n=0}^{N-1} \sin(\omega_k n)\cos(\omega_k n) = 0,\ k = 0, 1, 2, \ldots, N-1$$

More generally,

$$\sum_{n=0}^{N-1} x_e(n)x_o(n) = 0$$

for *any* even signal x_e and odd signal x_o in $\mathbf{C}^N$.

8.4 The Fourier Theorems

In this section the main Fourier theorems are stated and proved. It is no small matter how simple these theorems are in the DFT case relative to the other three cases (DTFT, Fourier transform, and Fourier series). When infinite summations or integrals are involved, the conditions for the existence of the Fourier transform can be quite difficult to characterize mathematically. Mathematicians have expended a considerable effort on such questions. By focusing primarily on the DFT case, we are able to study the essential concepts conveyed by the Fourier theorems without getting involved with mathematical difficulties.

8.4.1 Linearity

Theorem: For any $x, y \in \mathbf{C}^N$ and $\alpha, \beta \in \mathcal{C}$, the DFT satisfies

$$\boxed{\alpha x + \beta y \leftrightarrow \alpha X + \beta Y}$$

Thus, the DFT is a *linear operator*.

Proof:

$$\begin{aligned}
\mathrm{DFT}_k(\alpha x + \beta y) &\stackrel{\Delta}{=} \sum_{n=0}^{N-1} [\alpha x(n) + \beta y(n)] e^{-j2\pi nk/N} \\
&= \sum_{n=0}^{N-1} \alpha x(n) e^{-j2\pi nk/N} + \sum_{n=0}^{N-1} \beta y(n) e^{-j2\pi nk/N} \\
&= \alpha \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N} + \beta \sum_{n=0}^{N-1} y(n) e^{-j2\pi nk/N} \\
&\stackrel{\Delta}{=} \alpha X + \beta Y
\end{aligned}$$

8.4.2 Conjugation and Reversal

Theorem: For any $x \in \mathbf{C}^N$,

$$\boxed{\overline{x} \leftrightarrow \mathrm{FLIP}(\overline{X})}$$

Proof:

$$\begin{aligned}\mathrm{DFT}_k(\overline{x}) &\triangleq \sum_{n=0}^{N-1} \overline{x(n)} e^{-j2\pi nk/N} = \overline{\sum_{n=0}^{N-1} x(n) e^{j2\pi nk/N}} \\ &= \overline{\sum_{n=0}^{N-1} x(n) e^{-j2\pi n(-k)/N}} \triangleq \mathrm{FLIP}_k(\overline{X})\end{aligned}$$

Theorem: For any $x \in \mathbf{C}^N$,

$$\boxed{\mathrm{FLIP}(\overline{x}) \leftrightarrow \overline{X}}$$

Proof: Making the change of summation variable $m \triangleq N - n$, we get

$$\begin{aligned}\mathrm{DFT}_k(\mathrm{FLIP}(\overline{x})) &\triangleq \sum_{n=0}^{N-1} \overline{x(N-n)} e^{-j2\pi nk/N} = \sum_{m=N}^{1} \overline{x(m)} e^{-j2\pi(N-m)k/N} \\ &= \sum_{m=0}^{N-1} \overline{x(m)} e^{j2\pi mk/N} = \overline{\sum_{m=0}^{N-1} x(m) e^{-j2\pi mk/N}} \triangleq \overline{X(k)}\end{aligned}$$

Theorem: For any $x \in \mathbf{C}^N$,

$$\boxed{\mathrm{FLIP}(x) \leftrightarrow \mathrm{FLIP}(X)}$$

Proof:

$$\begin{aligned}\mathrm{DFT}_k[\mathrm{FLIP}(x)] &\triangleq \sum_{n=0}^{N-1} x(N-n) e^{-j2\pi nk/N} = \sum_{m=0}^{N-1} x(m) e^{-j2\pi(N-m)k/N} \\ &= \sum_{m=0}^{N-1} x(m) e^{j2\pi mk/N} \triangleq X(-k) \triangleq \mathrm{FLIP}_k(X)\end{aligned}$$

Corollary: For any $x \in \mathbf{R}^N$,

$$\boxed{\textsc{Flip}(x) \leftrightarrow \overline{X}} \quad (x \text{ real})$$

Proof: Picking up the previous proof at the third formula, remembering that x is real,

$$\sum_{n=0}^{N-1} x(n)e^{j2\pi nk/N} = \overline{\sum_{n=0}^{N-1} \overline{x(n)}e^{-j2\pi nk/N}} = \overline{\sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N}} \triangleq \overline{X(k)}$$

when $x(n)$ is real.

Thus, *conjugation in the frequency domain corresponds to reversal in the time domain*. Another way to say it is that *negating spectral phase flips the signal around backwards in time*.

Corollary: For any $x \in \mathbf{R}^N$,

$$\boxed{\textsc{Flip}(X) = \overline{X}} \quad (x \text{ real})$$

Proof: This follows from the previous two cases.

Definition: The property $X(-k) = \overline{X(k)}$ is called *Hermitian symmetry* or "conjugate symmetry." If $X(-k) = -\overline{X(k)}$, it may be called *skew-Hermitian*.

Another way to state the preceding corollary is

$$\boxed{x \in \mathbf{R}^N \leftrightarrow X \text{ is Hermitian}}$$

8.4.3 Symmetry

In the previous section, we found $\textsc{Flip}(X) = \overline{X}$ when x is real. This fact is of high practical importance. It says that the spectrum of every real signal is *Hermitian*. Due to this symmetry, we may discard all negative-frequency spectral samples of a real signal and regenerate them later if needed from the positive-frequency samples. Also, spectral plots of real signals are normally displayed only for positive frequencies; e.g., spectra of sampled signals are normally plotted over the range 0 Hz to $f_s/2$ Hz. On the other hand, the spectrum of a *complex* signal must be shown, in general, from $-f_s/2$ to $f_s/2$ (or from 0 to f_s), since the positive and negative frequency components of a complex signal are independent.

Theorem: If $x \in \mathbf{R}^N$, $\mathrm{re}\{X\}$ is *even* and $\mathrm{im}\{X\}$ is *odd*.

Proof: This follows immediately from the conjugate symmetry of X for real signals x.

Theorem: If $x \in \mathbf{R}^N$, $|X|$ is *even* and $\angle X$ is *odd*.

Proof: This follows immediately from the conjugate symmetry of X expressed in polar form $X(k) = |X(k)|\, e^{j\angle X(k)}$.

The conjugate symmetry of spectra of real signals is perhaps the most important symmetry theorem. However, there are a few more we can readily show.

Theorem: An even signal has an even transform:

$$\boxed{x \text{ even} \leftrightarrow X \text{ even}}$$

Proof: Express x in terms of its real and imaginary parts by $x \stackrel{\Delta}{=} x_r + jx_i$. Note that for a complex signal x to be even, both its real and imaginary parts must be even. Then

$$\begin{aligned}
X(k) &\stackrel{\Delta}{=} \sum_{n=0}^{N-1} x(n)e^{-j\omega_k n} \\
&= \sum_{n=0}^{N-1} [x_r(n) + jx_i(n)]\cos(\omega_k n) - j[x_r(n) + jx_i(n)]\sin(\omega_k n) \\
&= \sum_{n=0}^{N-1} [x_r(n)\cos(\omega_k n) + x_i(n)\sin(\omega_k n)] + j[x_i(n)\cos(\omega_k n) - x_r(n)\sin(\omega_k n)] \\
&= \sum_{n=0}^{N-1} \text{even} \cdot \text{even} + \underbrace{\sum_{n=0}^{N-1} \text{even} \cdot \text{odd}}_{\text{sums to 0}} + j \sum_{n=0}^{N-1} \text{even} \cdot \text{even} - j \underbrace{\sum_{n=0}^{N-1} \text{even} \cdot \text{odd}}_{\text{sums to 0}} \\
&= \sum_{n=0}^{N-1} \text{even} \cdot \text{even} + j \sum_{n=0}^{N-1} \text{even} \cdot \text{even} = \text{even} + j \cdot \text{even} = \text{even}
\end{aligned}$$

Theorem: A real even signal has a real even transform:

$$\boxed{x \text{ real and even} \leftrightarrow X \text{ real and even}}$$

Proof: This follows immediately from setting $x_i(n) = 0$ in the preceding proof and seeing that the DFT of a real and even function reduces to

a type of *cosine transform*[5],

$$X(k) = \sum_{n=0}^{N-1} x(n)\cos(\omega_k n),$$

or we can show it directly:

$$\begin{aligned} X(k) &\stackrel{\Delta}{=} \sum_{n=0}^{N-1} x(n)e^{-j\omega_k n} = \sum_{n=0}^{N-1} x(n)\cos(\omega_k n) + j\sum_{n=0}^{N-1} x(n)\sin(\omega_k n) \\ &= \sum_{n=0}^{N-1} x(n)\cos(\omega_k n) \qquad (\textstyle\sum \text{even} \cdot \text{odd} = \sum \text{odd} = 0) \\ &= \sum_{n=0}^{N-1} \text{even} \cdot \text{even} = \sum_{n=0}^{N-1} \text{even} = \text{even} \end{aligned}$$

Definition: A signal with a real spectrum (such as a real, even signal) is often called a *zero phase signal*. However, note that when the spectrum goes *negative* (which it can), the phase is really π, not 0. Nevertheless, it is common to call such signals "zero phase, " even though the phase switches between 0 and π at the zero-crossings of the spectrum. Such zero-crossings typically occur at low amplitude in practice, such as in the "sidelobes" of the DTFT of an FFT window.

8.4.4 Shift Theorem

Theorem: For any $x \in \mathbf{C}^N$ and any integer Δ,

$$\boxed{\mathrm{DFT}_k[\textsc{Shift}_\Delta(x)] = e^{-j\omega_k\Delta}X(k)}$$

[5]The discrete cosine transform (DCT) used often in applications is actually defined somewhat differently, but the basic principles are the same

Proof:

$$\begin{aligned}
\mathrm{DFT}_k[\textsc{Shift}_\Delta(x)] &\stackrel{\Delta}{=} \sum_{n=0}^{N-1} x(n-\Delta)e^{-j2\pi nk/N} \\
&= \sum_{m=-\Delta}^{N-1-\Delta} x(m)e^{-j2\pi(m+\Delta)k/N} \qquad (m \stackrel{\Delta}{=} n-\Delta) \\
&= \sum_{m=0}^{N-1} x(m)e^{-j2\pi mk/N}e^{-j2\pi k\Delta/N} \\
&= e^{-j2\pi\Delta k/N}\sum_{m=0}^{N-1} x(m)e^{-j2\pi mk/N} \\
&\stackrel{\Delta}{=} e^{-j\omega_k\Delta}X(k)
\end{aligned}$$

The shift theorem says that a *delay* in the time domain corresponds to a *linear phase term* in the frequency domain. More specifically, a delay of Δ samples in the time waveform corresponds to the linear phase term $e^{-j\omega_k\Delta}$ multiplying the spectrum, where $\omega_k \stackrel{\Delta}{=} 2\pi k/N$. (To consider ω_k as radians per second instead of radians per sample, just replace Δ by ΔT so that the delay is in seconds instead of samples.) Note that spectral magnitude is unaffected by a linear phase term. That is, $\left|e^{-j\omega_k\Delta}X(k)\right| = |X(k)|$.

Linear Phase Terms

The reason $e^{-j\omega_k\Delta}$ is called a *linear phase term* is that its phase is a linear function of frequency:

$$\angle e^{-j\omega_k\Delta} = -\Delta\cdot\omega_k$$

Thus, the slope of the phase versus radian frequency is $-\Delta$. In general, the *time delay in samples* equals *minus the slope of the linear phase term.* If we express the original spectrum in polar form as

$$X(k) = G(k)e^{j\Theta(k)},$$

where G and Θ are the magnitude and phase of X, respectively (both real), we can see that a linear phase term only modifies the spectral phase $\Theta(k)$:

$$e^{-j\omega_k\Delta}X(k) \stackrel{\Delta}{=} e^{-j\omega_k\Delta}G(k)e^{j\Theta(k)} = G(k)e^{j[\Theta(k)-\omega_k\Delta]}$$

where $\omega_k \stackrel{\Delta}{=} 2\pi k/N$. A positive time delay (waveform shift to the right) adds a *negatively sloped* linear phase to the original spectral phase. A negative time delay (waveform shift to the left) adds a *positively sloped* linear phase to the original spectral phase. If we seem to be belaboring this relationship, it is because it is one of the most useful in practice.

Definition: A signal is said to be *linear phase signal* if its phase is of the form

$$\Theta(\omega_k) = \pm\Delta \cdot \omega_k \pm \pi I(\omega_k)$$

where $I(\omega_k)$ is an *indicator function* which takes on the values 0 or 1.

Application of the Shift Theorem to FFT Windows

In practical spectrum analysis, we most often use the *fast Fourier transform*[6] (FFT) together with a *window function* $w(n), n = 0, 1, 2, \ldots, N-1$. Windows are normally positive ($w(n) > 0$), symmetric about their midpoint, and look pretty much like a "bell curve." A window multiplies the signal x being analyzed to form a *windowed signal* $x_w(n) = w(n)x(n)$, or $x_w = w \cdot x$, which is then analyzed using the FFT. The window serves to *taper* the data segment gracefully to zero, thus eliminating spectral distortions due to suddenly cutting off the signal in time. Windowing is thus appropriate when x is a short section of a longer signal.

Theorem: Real symmetric FFT windows are *linear phase*.

Proof: The midpoint of a symmetric signal can be translated to the time origin to create an even signal. As established previously, the DFT of a real even signal is real and even. By the shift theorem, the DFT of the original symmetric signal is a real even spectrum multiplied by a *linear phase term*. A spectrum whose phase is a linear function of frequency (with possible discontinuities of π radians), is *linear phase* by definition.

8.4.5 Convolution Theorem

Theorem: For any $x, y \in \mathbf{C}^N$,

$$\boxed{x * y \leftrightarrow X \cdot Y}$$

[6]The FFT is just a fast implementation of the DFT.

Proof:

$$\begin{aligned}
\mathrm{DFT}_k(x*y) &\triangleq \sum_{n=0}^{N-1}(x*y)_n e^{-j2\pi nk/N} \\
&\triangleq \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(m)y(n-m)e^{-j2\pi nk/N} \\
&= \sum_{m=0}^{N-1} x(m) \sum_{n=0}^{N-1} \underbrace{y(n-m)e^{-j2\pi nk/N}}_{e^{-j2\pi mk/N}Y(k)} \\
&= \left(\sum_{m=0}^{N-1} x(m)e^{-j2\pi mk/N}\right) Y(k) \quad \text{(by the Shift Theorem)} \\
&\triangleq X(k)Y(k)
\end{aligned}$$

This is perhaps the most important single Fourier theorem of all. It is the basis of a large number of applications of the FFT. Since the FFT provides a fast Fourier transform, it also provides *fast convolution*, thanks to the convolution theorem. It turns out that using the FFT to perform convolution is really more efficient in practice only for reasonably long convolutions, such as $N > 100$. For much longer convolutions, the savings become enormous compared with "direct" convolution. This happens because direct convolution requires on the order of N^2 operations (multiplications and additions), while FFT-based convolution requires on the order of $N \lg(N)$ operations.

The following simple Matlab example illustrates how much faster convolution can be performed using the FFT:

```
>> N = 1024;        % FFT much faster at this length
>> t = 0:N-1;       % [0,1,2,...,N-1]
>> h = exp(-t);     % filter impulse reponse = sampled exponential
>> H = fft(h);      % filter frequency response
>> x = ones(1,N);   % input = dc (any example will do)
>> t0 = clock; y = conv(x,h); t1 = etime(clock,t0);         % Direct
>> t0 = clock; y = ifft(fft(x) .* H); t2 = etime(clock,t0); % FFT
>> sprintf(['Elapsed time for direct convolution = %f sec\n',...
         'Elapsed time for FFT convolution = %f sec\n',...
         'Ratio = %f (Direct/FFT)'],t1,t2,t1/t2)
```

```
ans =

Elapsed time for direct convolution = 8.542129 sec
Elapsed time for FFT convolution = 0.075038 sec
Ratio = 113.837376 (Direct/FFT)
```

8.4.6 Dual of the Convolution Theorem

The *Dual*[7] of the Convolution Theorem says that *multiplication in the time domain* is *convolution in the frequency domain*:

Theorem:

$$\boxed{x \cdot y \leftrightarrow \frac{1}{N} X * Y}$$

Proof: The steps are the same as in the Convolution Theorem.

This theorem also bears on the use of *FFT windows*. It says that *windowing in the time domain* corresponds to *smoothing in the frequency domain*. That is, the spectrum of $w \cdot x$ is simply X filtered by W, or, $W * X$. This smoothing reduces *sidelobes* associated with the *rectangular window* (which is the window one gets implicitly when no window is explicitly used). FFT windows are covered in Music 420.

8.4.7 Correlation Theorem

Theorem: For all $x, y \in \mathbf{C}^N$,

$$\boxed{x \star y \leftrightarrow \overline{X} \cdot Y}$$

Proof:

$$\begin{aligned}
(x \star y)_n &\stackrel{\Delta}{=} \sum_{m=0}^{N-1} \overline{x(m)} y(n+m) \\
&= \sum_{m=0}^{N-1} \overline{x(-m)} y(n-m) \qquad (m \leftarrow -m) \\
&= (\textsc{Flip}(\overline{x}) * y)_n \\
&\leftrightarrow \overline{X} \cdot Y
\end{aligned}$$

[7] In general, the *dual* of any Fourier operation is obtained by interchanging time and frequency.

where the last step follows from the Convolution Theorem and the result $\textsc{Flip}(\overline{x}) \leftrightarrow \overline{X}$ from the section on Conjugation and Reversal.

8.4.8 Power Theorem

Theorem: For all $x, y \in \mathbf{C}^N$,

$$\boxed{\langle x, y\rangle = \frac{1}{N}\langle X, Y\rangle}$$

Proof:

$$\begin{aligned}\langle x, y\rangle &\stackrel{\Delta}{=} \sum_{n=0}^{N-1} x(n)\overline{y(n)} = (y \star x)_0 = \mathrm{DFT}_0^{-1}(\overline{Y}\cdot X)\\ &= \frac{1}{N}\sum_{k=0}^{N-1} X(k)\overline{Y(k)} \stackrel{\Delta}{=} \frac{1}{N}\langle X, Y\rangle\end{aligned}$$

Note that the power theorem would be more elegant ($\langle x, y\rangle = \langle X, Y\rangle$) if the DFT were defined as the coefficient of projection onto the *normalized DFT sinusoid* $\tilde{s}_k(n) \stackrel{\Delta}{=} s_k(n)/\sqrt{N}$.

8.4.9 Rayleigh Energy Theorem (Parseval's Theorem)

Theorem: For any $x \in \mathbf{C}^N$,

$$\boxed{\| x \|^2 = \frac{1}{N}\| X \|^2}$$

I.e.,

$$\boxed{\sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |X(k)|^2}$$

Proof: This is a special case of the Power Theorem.

Note that again the relationship would be cleaner ($\| x \| = \| X \|$) if we were using the *normalized DFT*.

8.4.10 Stretch Theorem (Repeat Theorem)

Theorem: For all $x \in \mathbf{C}^N$,

$$\boxed{\textsc{Stretch}_L(x) \leftrightarrow \textsc{Repeat}_L(X)}$$

Proof: Recall the stretch operator:

$$\textsc{Stretch}_{L,m}(x) \triangleq \begin{cases} x(m/L), & m/L = \text{integer} \\ 0, & m/L \neq \text{integer} \end{cases}$$

Let $y \triangleq \textsc{Stretch}_L(x)$, where $y \in \mathcal{C}^M$, $M = LN$. Also define the new denser frequency grid associated with length M by $\omega'_k \triangleq 2\pi k/M$, with $\omega_k = 2\pi k/N$ as usual. Then

$$Y(k) \triangleq \sum_{m=0}^{M-1} y(m)e^{-j\omega'_k m} = \sum_{n=0}^{N-1} x(n)e^{-j\omega'_k nL} \qquad (n \triangleq m/L)$$

But

$$\omega'_k L \triangleq \frac{2\pi k}{M}L = \frac{2\pi k}{N} = \omega_k$$

Thus, $Y(k) = X(k)$, and by the modulo indexing of X, L copies of X are generated as k goes from 0 to $M-1 = LN-1$.

8.4.11 Downsampling Theorem (Aliasing Theorem)

Theorem: For all $x \in \mathbf{C}^N$,

$$\boxed{\textsc{Select}_L(x) \leftrightarrow \frac{1}{L}\textsc{Alias}_L(X)}$$

Proof: Let $k' \in [0, M-1]$ denote the frequency index in the aliased spectrum, and let $Y(k') \triangleq \textsc{Alias}_{L,k'}(X)$. Then Y is length $M = N/L$, where L is the downsampling factor. We have

$$\begin{aligned} Y(k') &\triangleq \textsc{Alias}_{L,k'}(X) \triangleq \sum_{l=0}^{L-1} Y(k'+lM), \quad k' = 0,1,2,\ldots,M-1 \\ &\triangleq \sum_{l=0}^{L-1}\sum_{n=0}^{N-1} x(n)e^{-j2\pi(k'+lM)n/N} \\ &\triangleq \sum_{n=0}^{N-1} x(n)e^{-j2\pi k'n/N}\sum_{l=0}^{L-1} e^{-j2\pi lnM/N} \end{aligned}$$

Since $M/N = L$, the sum over l becomes

$$\sum_{l=0}^{L-1} e^{-j2\pi lMn/N} = \sum_{l=0}^{L-1} \left[e^{-j2\pi n/L}\right]^l = \frac{1-e^{-j2\pi n}}{1-e^{-j2\pi n/L}} = \left\{ \begin{array}{ll} L, & n = 0 \mod L \\ 0, & n \neq 0 \mod L \end{array} \right.$$

using the closed form expression for a geometric series derived earlier. We see that the sum over L effectively *samples* x every L samples. This can be expressed in the previous formula by defining $m \stackrel{\Delta}{=} n/L$ which ranges only over the nonzero samples:

$$\begin{aligned} \textsc{Alias}_{L,k'}(X) &= \sum_{n=0}^{N-1} x(n) e^{-j2\pi k' n/N} \sum_{l=0}^{L-1} e^{-j2\pi ln/L} \\ &= L \sum_{m=0}^{N/L-1} x(mL) e^{-j2\pi k'(mL)/N} \qquad (m \stackrel{\Delta}{=} n/L) \\ &\stackrel{\Delta}{=} L \sum_{m=0}^{M-1} x(mL) e^{-j2\pi k' m/M} \stackrel{\Delta}{=} L \sum_{m=0}^{M-1} \textsc{Select}_{L,m}(x) e^{-j2\pi k' m/M} \\ &\stackrel{\Delta}{=} L \cdot \mathrm{DFT}_{k'}(\textsc{Select}_L(x)) \end{aligned}$$

Since the above derivation also works in reverse, the theorem is proved.

Here is an illustration of the Downsampling Theorem in Matlab:

```
>> N=4;
>> x = 1:N;
>> X = fft(x);
>> x2 = x(1:2:N);
>> fft(x2)                          % FFT(Decimate(x,2))

ans =

     4    -2

>> (X(1:N/2) + X(N/2 + 1:N))/2     % (1/2) Alias(X,2)

ans =

    4.0000   -2.0000
```

An illustration of aliasing in the frequency domain is shown in Fig. 8.10.

8.4.12 Zero Padding Theorem

A fundamental tool in practical spectrum analysis is *zero padding*. This theorem shows that zero padding in the time domain corresponds to ideal interpolation in the frequency domain:

Let $x \in \mathbf{C}^N$ and define $y = \textsc{ZeroPad}_M(x)$. Then $y \in \mathcal{C}^M$ with $M \geq N$. Denote the original frequency index by k, where $\omega_k \triangleq 2\pi k/N$ and the new frequency index by k', where $\omega_{k'} \triangleq 2\pi k'/M$.

Definition: The *ideal bandlimited interpolation* of a spectrum $X(\omega_k) \triangleq \mathrm{DFT}_k(x)$, $x \in \mathbf{C}^N$, to an arbitrary new frequency $\omega \in [-\pi, \pi)$ is defined as

$$X(\omega) \triangleq \sum_{n=0}^{N-1} x(n) e^{-j\omega n}$$

Note that this is just the definition of the DFT with ω_k replaced by ω. That is, the spectrum is interpolated by projecting onto the new sinusoid exactly as if it were a DFT sinusoid. This makes the most sense when x is assumed to be N samples of a *time-limited* signal. That is, if the signal really is zero outside of the time interval $[0, N-1]$, then the inner product between it and any sinusoid will be exactly as in the equation above. Thus, for time limited signals, this kind of interpolation is ideal.

Definition: The *interpolation operator* interpolates a signal by an integer factor L. That is,

$$\textsc{Interp}_{L,k'}(X) \triangleq X(\omega_{k'}), \quad \omega_{k'} = 2\pi k'/M,\ k' = 0, 1, 2, \ldots, M-1,\ M \triangleq LN$$

Since $X(\omega_k) \triangleq \mathrm{DFT}_{N,k}(x)$ is initially only defined over the N roots of unity, while $X(\omega_{k'})$ is defined over $M = LN$ roots of unity, we define $X(\omega_{k'})$ for $\omega_{k'} \neq \omega_k$ by ideal bandlimited interpolation.

Theorem: For any $x \in \mathbf{C}^N$

$$\boxed{\textsc{ZeroPad}_{LN}(x) \leftrightarrow \textsc{Interp}_L(X)}$$

Proof: Let $M = LN$ with $L \geq 1$. Then

$$\mathrm{DFT}_{M,k'}(\textsc{ZeroPad}_M(x)) = \sum_{m=0}^{N-1} x(m) e^{-j2\pi mk'/M} \triangleq X(\omega_{k'}) = \textsc{Interp}_L(X)$$

8.4.13 Bandlimited Interpolation in Time

The *dual* of the Zero-Padding Theorem states formally that *zero padding in the frequency domain* corresponds to *ideal bandlimited interpolation in the time domain*. However, we have not precisely defined ideal bandlimited interpolation in the time domain. Therefore, we'll let the dual of the Zero-Padding Theorem provide its definition:

Definition: For all $x \in \mathbf{C}^N$ and any integer $L \geq 1$,

$$\boxed{\textsc{Interp}_L(x) \stackrel{\Delta}{=} \mathrm{IDFT}(\textsc{ZeroPad}_{LN}(X))}$$

where the zero-padding is of the frequency-domain type, as described earlier and illustrated in Fig. 8.5.

It is instructive to interpret the Interpolation Theorem in terms of the Stretch Theorem $\textsc{Stretch}_L(x) \leftrightarrow \textsc{Repeat}_L(X)$. To do this, it is convenient to define a "zero-centered rectangular window" operator:

Definition: For any $X \in \mathbf{C}^N$ and any odd integer $M < N$ we define the *length M even rectangular windowing operation* by

$$\textsc{RectWin}_{M,k}(X) \stackrel{\Delta}{=} \begin{cases} X(k), & -\frac{M-1}{2} \leq k \leq \frac{M-1}{2} \\ 0, & \frac{M+1}{2} \leq |k| \leq \frac{N}{2} \end{cases}$$

Thus, the "zero-phase rectangular window," when applied to a spectrum X, sets the spectrum to zero everywhere outside a zero-centered interval of M samples. Note that $\textsc{RectWin}_M(X)$ is the *ideal lowpass filtering operation in the frequency domain*, where the lowpass "cut-off frequency" in radians per sample is $\omega_c = 2\pi[(M-1)/2]/N$. With this we can efficiently show the basic theorem of *ideal bandlimited interpolation*:

Theorem: For $x \in \mathbf{C}^N$,

$$\boxed{\textsc{Interp}_L(x) = \mathrm{IDFT}(\textsc{RectWin}_N(\mathrm{DFT}(\textsc{Stretch}_L(x))))}$$

In other words, ideal bandlimited interpolation of x by the factor L may be carried out by first stretching x by the factor L (i.e., inserting $L-1$ zeros between adjacent samples of x), taking the DFT, applying the ideal lowpass filter, and performing the inverse DFT.

Proof: First, recall that $\textsc{Stretch}_L(x) \leftrightarrow \textsc{Repeat}_L(X)$, that is, stretching a signal by the factor L gives a new signal $y = \textsc{Stretch}_L(x)$ which has a spectral grid L times the density of X, and the spectrum Y contains L copies of X repeated around the unit circle. The "baseband copy"

of X can be defined as the width N sequence centered about frequency zero. Therefore, if we can use an "ideal filter" to "pass" the baseband spectral copy and zero out all others, we can convert $\textsc{Repeat}_L(X)$ to $\textsc{ZeroPad}_{LN}(X)$. I.e.,

$$\textsc{RectWin}_N(\textsc{Repeat}_L(X)) = \textsc{ZeroPad}_{LN}(X) \leftrightarrow \textsc{Interp}_L(x)$$

where the last step is by definition of time-domain ideal bandlimited interpolation.

Note that the definition of ideal bandlimited time-domain interpolation in this section is only really ideal for signals which are *periodic* in N samples. To see this, consider that the rectangular windowing operation in the frequency domain corresponds to *cyclic convolution* in the time domain,[8] and cyclic convolution is only the same as acyclic convolution when one of the signals is truly periodic in N samples. Since all spectra $X \in \mathbf{C}^N$ are truly periodic in N samples, there is no problem with the definition of ideal spectral interpolation used in connection with the Zero-Padding Theorem. However, for a more practical definition of ideal time-domain interpolation, we should use instead the dual of the Zero-Padding Theorem for the DTFT case. Nevertheless, for signals which are *exactly* periodic in N samples (a rare situation), the present definition is ideal.

8.5 Conclusions

For the student interested in pursuing further the topics of this reader, see [18, 19].

8.6 Acknowledgement

Thanks to Craig Stuart Sapp (craig@ccrma.stanford.edu) for contributing Figures 8.2, 8.3, 8.4, 8.6, 8.7, 8.8, 8.9, and 8.10.

[8] The inverse DFT of the rectangular window here is an aliased, discrete-time counterpart of the *sinc function* which is defined as "$\sin(\pi t)/\pi t$." The sinc function is covered in Music 420.

8.7 Appendix A: Linear Time-Invariant Filters and Convolution

A reason for the importance of convolution is that *every linear time-invariant system*[9] *can be represented by a convolution*. Thus, in the convolution equation

$$y = h * x$$

we may interpret x as the *input* signal to a filter, y as the *output* signal, and h as the *digital filter*, as shown in Fig. 8.11.

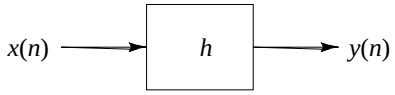

Figure 8.11: The filter interpretation of convolution.

The *impulse* or "unit pulse" signal is defined by

$$\delta(n) \stackrel{\Delta}{=} \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

For example, for $N = 4$, $\delta = [1, 0, 0, 0]$. The impulse signal is the *identity element* under convolution, since

$$(x * \delta)_n \stackrel{\Delta}{=} \sum_{m=0}^{N-1} \delta(m)x(n-m) = x(n)$$

If we set $x = \delta$ in the filter equation above, we get

$$y = h * x = h * \delta = h$$

Thus, h is the *impulse response* of the filter.

It turns out in general that every linear time-invariant (LTI) system (filter) is completely described by its impulse response. No matter what the LTI system is, we can give it an impulse, record what comes out, call it $h(n)$, and implement the system by convolving the input signal x with the impulse response h. In other words, every LTI system has a *convolution representation* in terms of its impulse response.

[9] We use the term "system" interchangeably with "filter."

8.7.1 LTI Filters and the Convolution Theorem

Definition: The *frequency response* of an LTI filter is defined as the Fourier transform of its impulse response. In particular, for finite, discrete-time signals $h \in \mathbf{C}^N$, the sampled frequency response is defined as

$$H(\omega_k) \triangleq DFT_k(h)$$

The complete frequency response is defined using the DTFT, i.e.,

$$H(\omega) \triangleq \mathrm{DTFT}_\omega(\textsc{ZeroPad}_\infty(h)) \triangleq \sum_{n=0}^{N-1} h(n)e^{-j\omega n}$$

where we used the fact that $h(n)$ is zero for $n < 0$ and $n > N-1$ to truncate the summation limits. Thus, the infinitely zero-padded DTFT can be obtained from the DFT by simply replacing ω_k by ω. In principle, the continuous frequency response $H(\omega)$ is being obtained using "time-limited interpolation in the frequency domain" based on the samples $H(\omega_k)$. This interpolation is possible only when the frequency samples $H(\omega_k)$ are sufficiently dense: for a length N finite-impulse-response (FIR) filter h, we require at least N samples around the unit circle (length N DFT) in order that $H(\omega)$ be sufficiently well sampled in the frequency domain. This is of course the dual of the usual sampling rate requirement in the time domain.[10]

Definition: The *amplitude response* of a filter is defined as the *magnitude* of the frequency response

$$G(k) \triangleq |H(\omega_k)|$$

From the convolution theorem, we can see that the amplitude response $G(k)$ is the *gain* of the filter at frequency ω_k, since

$$|Y(k)| = |H(\omega_k)X(k)| = G(k)\,|X(k)|$$

[10]Note that we normally say the sampling rate in the time domain must be higher than *twice* the highest frequency in the frequency domain. From the point of view of this reader, however, we may say instead that sampling rate in the time domain must be greater than the *full spectral bandwidth* of the signal, including both positive and negative frequencies. From this simplified point of view, "sampling rate = bandwidth supported."

Definition: The *phase response* of a filter is defined as the *phase* of the frequency response

$$\Theta(k) \stackrel{\Delta}{=} \angle H(\omega_k)$$

From the convolution theorem, we can see that the phase response $\Theta(k)$ is the phase-shift added by the filter to an input sinusoidal component at frequency ω_k, since

$$\angle Y(k) = \angle[H(\omega_k)X(k)] = \angle H(\omega_k) + \angle X(k)$$

The subject of this section is developed in detail in [4].

8.8 Appendix B: Statistical Signal Processing

The *correlation operator* defined above plays a major role in statistical signal processing. This section gives a short introduction to some of the most commonly used elements. The student interested in mastering the concepts introduced briefly below may consider taking EE 278 in the Electrical Engineering Department. For further reading, see [12, 20].

8.8.1 Cross-Correlation

Definition: The *circular cross-correlation* of two signals x and y in $\mathbf{C}^N$ may be defined by

$$\boxed{r_{xy}(l) \stackrel{\Delta}{=} \frac{1}{N}(x \star y)(l) \stackrel{\Delta}{=} \frac{1}{N}\sum_{n=0}^{N-1} \overline{x(n)}y(n+l),\ l = 0, 1, 2, \ldots, N-1} \quad \text{(cross-correlation)}$$

(Note carefully above that "l" is an integer variable, not the constant 1.) The term "cross-correlation" comes from *statistics*, and what we have defined here is more properly called the "sample cross-correlation," i.e., it is an *estimator* of the true cross-correlation which is a statistical property of the signal itself. The estimator works by *averaging lagged products* $\overline{x(n)}y(n+l)$. The true statistical cross-correlation is the so-called *expected value* of the lagged products in random signals x and y, which may be denoted $\mathcal{E}\{x(n)y(n+l)\}$. In principle, the expected value must be computed by averaging $(n)y(n+l)$ over many *realizations* of the stochastic process x and y. That is, for each "roll of the dice" we obtain $x(\cdot)$ and $y(\cdot)$ for *all time*, and we can average $x(n)y(n+l)$ across all

realizations to estimate the expected value of $x(n)y(n+l)$. This is called an "ensemble average" across realizations of a stochastic process. If the signals are *stationary* (which primarily means their statistics are *time-invariant*), then we may *average across time* to estimate the expected value. In other words, for stationary noise-like signals, *time averages equal ensemble averages*. The above definition of the sample cross-correlation is only valid for stationary stochastic processes.

The DFT of the cross-correlation is called the *cross-spectral density*, or "cross-power spectrum," or even simply "cross-spectrum."

Normally in practice we are interested in estimating the true cross-correlation between two signals, not the circular cross-correlation which results naturally in a DFT setting. For this, we may define instead the *unbiased cross-correlation*

$$\boxed{\hat{r}_{xy}(l) \triangleq \frac{1}{N-1-l}\sum_{n=0}^{N-l}\overline{x(n)}y(n+l),\ l=0,1,2,\ldots,L-1}$$

where we chose $L << N$ (e.g. $L=\sqrt{N}$) in order to have enough lagged products at the highest lag that a reasonably accurate average is obtained. The term "unbiased" refers to the fact that we are dividing the sum by $N-l$ rather than N.

Note that instead of first estimating the cross-correlation between signals x and y and then taking the DFT to estimate the cross-spectral density, we may instead compute the sample cross-correlation for each block of a signal, take the DFT of each, and *average the DFTs* to form a final cross-spectrum estimate. This is called the *periodogram method* of spectral estimation.

8.8.2 Applications of Cross-Correlation

In this section, two applications of cross-correlation are outlined.

Matched Filtering

The cross-correlation function is used extensively in *pattern recognition* and *signal detection*. We know that projecting one signal onto another is a means of measuring how much of the second signal is present in the first. This can be used to "detect" the presence of known signals as components of more complicated signals. As a simple example, suppose we record $x(n)$

which we think consists of a signal $s(n)$ which we are looking for plus some additive measurement noise $e(n)$. Then the projection of x onto s is

$$\mathcal{P}_s(x) = \mathcal{P}_s(s) + \mathcal{P}_s(e) \approx s$$

since the projection of any specific signal s onto random, zero-mean noise is close to zero. Another term for this process is called *matched filtering*. The impulse response of the "matched filter" for a signal x is given by $\textsc{Flip}(x)$. By time reversing x, we transform the convolution implemented by filtering into a cross-correlation operation.

FIR System Identification

Estimating an impulse response from input-output measurements is called *system identification*, and a large literature exists on this topic [21].

Cross-correlation can be used to compute the impulse response $h(n)$ of a filter from the cross-correlation of its input and output signals $x(n)$ and $y = h * x$, respectively.

To see this, note that, by the correlation theorem,

$$x \star y \leftrightarrow \overline{X} \cdot Y = \overline{X} \cdot (H \cdot X) = H \cdot |X|^2$$

Therefore, the frequency response is given by the input-output cross-spectrum divided by the input power spectrum:

$$H = \frac{\overline{X} \cdot Y}{|X|^2}$$

In terms of the cross-spectral density and the input power-spectral density (which can be estimated by *averaging* $\overline{X} \cdot Y$ and $|X|^2$, respectively), this relation can be written as

$$H(\omega) = \frac{R_{xy}(\omega)}{R_{xx}(\omega)}$$

Multiplying the above equation by $R_{xx}(\omega)$ and taking the inverse DTFT yields the time-domain relation

$$h * r_{xx} = r_{xy}$$

where r_{xy} can also be written as $x \star y$. In words, the cross-correlation of the filter input and output is equal to the filter's impulse response convolved with the autocorrelation of the input signal.

A Matlab program illustrating these relationships is listed in Fig. 8.12.

```
% sidex.m - Demonstration of the use of FFT cross-
% correlation to compute the impulse response
% of a filter given its input and output.
% This is called "FIR system identification".

Nx = 32; % input signal length
Nh = 10; % filter length Ny = Nx+Nh-1;
% max output signal length
% FFT size to accommodate cross-correlation:
Nfft = 2^nextpow2(Nx+Ny-1); % FFT wants power of 2

x = rand(1,Nx); % input signal = noise
%x = 1:Nx;  % input signal = ramp
h = [1:Nh];  % the filter
xzp = [x,zeros(1,Nfft-Nx)]; % zero-padded input
yzp = filter(h,1,xzp); % apply the filter
X = fft(xzp);   % input spectrum
Y = fft(yzp);   % output spectrum
Rxx = conj(X) .* X; % energy spectrum of x
Rxy = conj(X) .* Y; % cross-energy spectrum
Hxy = Rxy ./ Rxx;   % should be the freq. response
hxy = ifft(Hxy);    % should be the imp. response

hxy(1:Nh)      % print estimated impulse response
freqz(hxy,1,Nfft);  % plot estimated freq response

err = norm(hxy - [h,zeros(1,Nfft-Nh)])/norm(h);
disp(sprintf(['Impulse Response Error = ',...
'%0.14f%%'],100*err));

err = norm(Hxy-fft([h,zeros(1,Nfft-Nh)]))/norm(h);
disp(sprintf('Frequency Response Error = ',...
'%0.14f%%'],100*err));
```

Figure 8.12: FIR system identification example in Matlab.

8.8.3 Autocorrelation

The cross-correlation of a signal with itself gives the *autocorrelation function*

$$\boxed{r_x(l) \stackrel{\Delta}{=} \frac{1}{N}(x \star x)(l) \stackrel{\Delta}{=} \frac{1}{N}\sum_{n=0}^{N-1} \overline{x(n)}x(n+l)}$$

The autocorrelation function is Hermitian:

$$r_x(-l) = \overline{r_x(l)}$$

When x is real, its autocorrelation is *symmetric*. More specifically, it is *real and even*.

As in the case of cross-correlation, we can form an unbiased sample autocorrelation as

$$\boxed{\hat{r}_x(l) \stackrel{\Delta}{=} \frac{1}{N-l}\sum_{n=0}^{N-1} \overline{x(n)}x(n+l),\ l = 0, 1, 2, \ldots, L-1}$$

The DFT of the autocorrelation function $r_x(n)$ is called the *power spectral density* (PSD), or *power spectrum*, and is often denoted

$$R_x(k) \stackrel{\Delta}{=} \mathrm{DFT}_k(r_x)$$

The true PSD of a "stationary stochastic process" is the Fourier transform of the true autocorrelation function, and therefore the definition above provides only a sample *estimate* of the PSD.

Periodogram Method for Power Spectrum Estimation

As in the case of the cross-spectrum, we may use the *periodogram method* for computing the power spectrum estimate. That is, we may estimate the power spectrum as the average of the DFTs of many sample autocorrelations which are computed block by block in a long signal, rather than taking one DFT of a single autocorrelation estimate based on all the data we have. By the Correlation Theorem, this is the same as *averaging squared-magnitude DFTs of the signal blocks themselves*. Let x_m denote the mth block of the signal x, and let M denote the number of blocks. Then the PSD estimate is given by

$$\hat{R}_x(k) = \frac{1}{M}\left|DFT_k(x_m)\right|^2$$

However, note that $|X_m|^2 \leftrightarrow x \star x$ which is *circular* correlation. To avoid this, we use *zero padding* in the time domain, i.e., we replace x_m above by $[x_m, 0, \ldots, 0]$. However, note that although the "wrap-around problem" is fixed, the estimator is still *biased*. To repair this, we can use a *triangular window* (also called a "Bartlett window") to apply the weighting needed to remove the bias.

For real signals, the autocorrelation is real and even, and therefore the power spectral density is real and even for all real signals. The PSD $R_x(\omega)$ can interpreted as a measure of the relative probability that the signal contains energy at frequency ω. Essentially, however, it is the long-term average energy density vs. frequency in the random process $x(n)$.

At lag zero, the autocorrelation function reduces to the *average power* (root mean square) which we defined earlier:

$$r_x(0) \triangleq \frac{1}{N}\sum_{m=0}^{N-1} |x(m)|^2 \triangleq \mathcal{P}_x^2$$

Replacing "correlation" with "covariance" in the above definitions gives the corresponding zero-mean versions. For example, the *cross-covariance* is defined as

$$\boxed{c_{xy}(n) \triangleq \frac{1}{N}\sum_{m=0}^{N-1} \overline{[x(m)-\mu_x]}[y(m+n)-\mu_y]}$$

We also have that $c_x(0)$ equals the *variance* of the signal x:

$$c_x(0) \triangleq \frac{1}{N}\sum_{m=0}^{N-1} |x(m)-\mu_x|^2 \triangleq \sigma_x^2$$

8.8.4 Coherence

A function related to cross-correlation is the *coherence function* $\Gamma_{xy}(\omega)$, defined in terms of power spectral densities and the cross-spectral density by

$$\Gamma_{xy}(k) \triangleq \frac{R_{xy}(k)}{\sqrt{R_x(k)R_y(k)}}$$

In practice, these quantities can be *estimated* by *averaging* $\overline{X(k)}Y(k)$, $|X(k)|^2$ and $|Y(k)|^2$ over successive signal blocks. Let $\{\cdot\}$ denote time

averaging. Then an estimate of the coherence, the *sample coherence function* $\hat{\Gamma}_{xy}(k)$, may be defined by

$$\hat{\Gamma}_{xy}(k) \triangleq \frac{\left\{\overline{X(k)}Y(k)\right\}}{\sqrt{\left\{|X(k)|^2\right\} \cdot \left\{|Y(k)|^2\right\}}}$$

The magnitude-squared coherence $|\Gamma_{xy}(k)|^2$ is a real function between 0 and 1 which gives a measure of correlation between x and y at each frequency (DFT bin number k). For example, imagine that y is produced from x via an LTI filtering operation:

$$y = h * x \quad \Longrightarrow \quad Y(k) = H(k)X(k)$$

Then the coherence function is

$$\hat{\Gamma}_{xy}(k) \triangleq \frac{\overline{X(k)}Y(k)}{|X(k)| \cdot |Y(k)|} = \frac{\overline{X(k)}H(k)X(k)}{|X(k)|\,|H(k)X(k)|} = \frac{|X(k)|^2 H(k)}{|X(k)|^2\,|H(k)|} = \frac{H(k)}{|H(k)|}$$

and the magnitude-squared coherence function is simply

$$\left|\hat{\Gamma}_{xy}(k)\right| = \left|\frac{H(k)}{|H(k)|}\right| = 1$$

On the other hand, if x and y are uncorrelated noise processes, the coherence converges to zero.

A common use for the coherence function is in the validation of input/output data collected in an acoustics experiment for purposes of *system identification*. For example, $x(n)$ might be a known signal which is input to an unknown system, such as a reverberant room, say, and $y(n)$ is the recorded response of the room. Ideally, the coherence should be 1 at all frequencies. However, if the microphone is situated at a *null* in the room response for some frequency, it may record mostly noise at that frequency. This will be indicated in the measured coherence by a significant dip below 1.

8.9 Appendix C: The Similarity Theorem

The *similarity theorem* is fundamentally restricted to the *continuous*-time case. It says that if you "stretch" a signal by the factor α in the time

domain, you "squeeze" its Fourier transform by the same factor in the frequency domain. This is such a fundamental Fourier relationship, that we include it here rather than leave it out as a non-DFT result.

The closest we came to the similarity theorem among the DFT theorems was the the Interpolation Theorem. We found that "stretching" a *discrete-time* signal by the integer factor α (filling in between samples with zeros) corresponded to the spectrum being *repeated* α times around the unit circle. As a result, the "baseband" copy of the spectrum "shrinks" in width (relative to 2π) by the factor α. Similarly, stretching a signal using *interpolation* (instead of zero-fill) corresponded to the repeated spectrum with all spurious spectral copies zeroed out. The spectrum of the interpolated signal can therefore be seen as having been stretched by the inverse of the time-domain stretch factor. In summary, the Interpolation DFT Theorem can be viewed as the discrete-time counterpart of the similarity Fourier Transform (FT) theorem.

Theorem: For all continuous-time functions $x(t)$ possessing a Fourier transform,

$$\boxed{\textsc{Stretch}_\alpha(x) \leftrightarrow \frac{1}{|\alpha|}\textsc{Stretch}_{(1/\alpha)}(X)}$$

where

$$\textsc{Stretch}_{\alpha,t}(x) \stackrel{\Delta}{=} x(\alpha t)$$

and α is any nonzero real number (the abscissa scaling factor).

Proof:

$$\begin{aligned}
\mathrm{FT}_\omega(\textsc{Stretch}_\alpha(x)) &\stackrel{\Delta}{=} \int_{-\infty}^{\infty} x(\alpha t)e^{-j\omega t}dt = \int_{-\infty}^{\infty} x(\tau)e^{-j\omega(\tau/\alpha)}d(\tau/\alpha) \\
&= \frac{1}{|\alpha|}\int_{-\infty}^{\infty} x(\tau)e^{-j(\omega/\alpha)\tau}d\tau \\
&\stackrel{\Delta}{=} \frac{1}{|\alpha|}X\left(\frac{\omega}{\alpha}\right)
\end{aligned}$$

The absolute value appears above because, when $\alpha < 0$, $d(\tau/\alpha) < 0$, which brings out a minus sign in front of the integral from $-\infty$ *to* ∞.

Chapter 9

Example Applications of the DFT

This chapter goes through some practical examples of FFT analysis in Matlab. The various Fourier theorems provide a "thinking vocabulary" for understanding elements of spectral analysis.

9.1 Spectrum Analysis of a Sinusoid: Windowing, Zero-Padding, and the FFT

The examples below give a progression from the most simplistic analysis up to a proper practical treatment. Careful study of these examples will teach you a lot about how spectrum analysis is carried out on real data, and provide opportunities to see the Fourier theorems in action.

9.1.1 Example 1: FFT of a Simple Sinusoid

Our first example is an FFT of the simple sinusoid

$$x(n) = \cos(\omega_x n T)$$

where we choose $\omega_x = 2\pi(f_s/4)$ (frequency $f_s/4$) and $T = 1$ (sampling rate set to 1). Since we're using the FFT, the signal length N must be a power of 2. Here is the Matlab code:

```
echo on; hold off; diary off;
```

```
% !/bin/rm -f examples.dia; diary examples.dia    % For session log
% !mkdirs eps                                     % For figures

% Example 1: FFT of a DFT sinusoid

% Parameters:
N = 64;               % Must be a power of two
T = 1;                % Set sampling rate to 1
f = 0.25;             % Sinusoidal frequency in cycles per sample
A = 1;                % Sinusoidal amplitude
phi = 0;              % Sinusoidal phase
n = [0:N-1];
x = cos(2*pi*n*f*T);  % Signal to analyze
X = fft(x);           % Spectrum
```

Let's plot the time data and magnitude spectrum:

```
% Plot time data
figure(1);
subplot(3,1,1);
plot(n,x,'*');
ni = [0:.1:N-1];      % Interpolated time axis
hold on;
plot(ni,cos(2*pi*ni*f*T),'-');
title('Sinusoid Sampled at 1/4 the Sampling Rate');
xlabel('Time (samples)'); ylabel('Amplitude');
text(-8,1,'a)');

% Plot spectral magnitude
magX = abs(X);
fn = [0:1.0/N:1-1.0/N];  % Normalized frequency axis
subplot(3,1,2);
stem(fn,magX)
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (Linear)');
text(-.11,40,'b)');

% Same thing on a dB scale
spec = 20*log10(magX);  % Spectral magnitude in dB
subplot(3,1,3);
```

```
plot(fn,spec);
axis([0 1 -350 50]);
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)');
text(-.11,50,'c)');
print -deps eps/example1.eps; hold off;
```

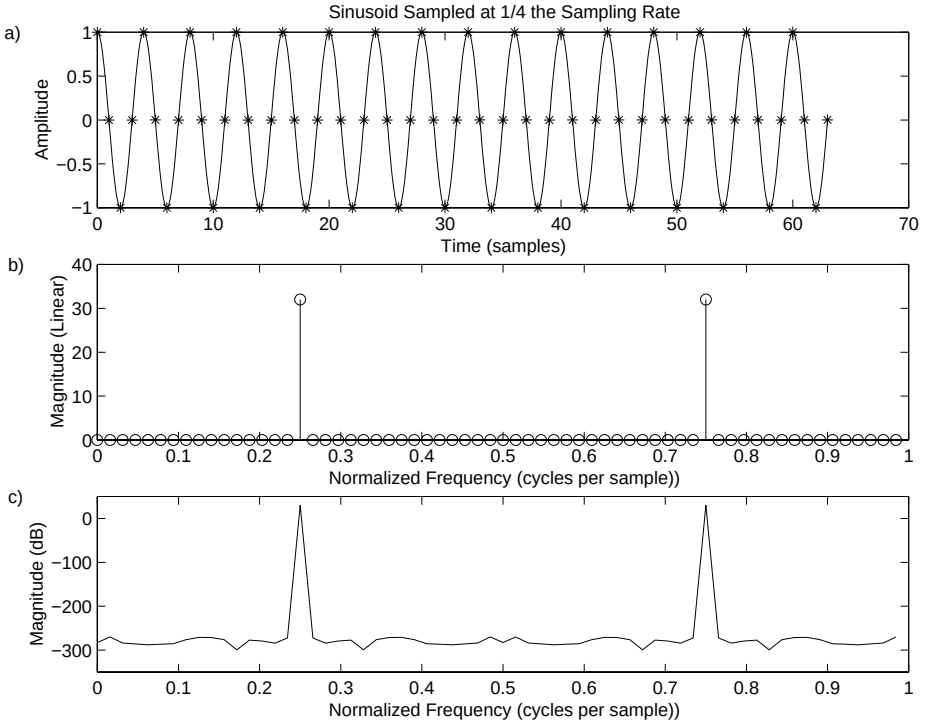

Figure 9.1: Sampled sinusoid at $f = f_s/4$. a) Time waveform. b) Magnitude spectrum. c) DB magnitude spectrum.

The results are shown in Fig. 9.1. The time-domain signal is shown in Fig. 9.1a, both in pseudo-continuous and sampled form. In Fig. 9.1b, we see two peaks in the magnitude spectrum, each at magnitude 32 on a linear scale, located at normalized frequencies $f = 0.25$ and $f = 0.75 = -0.25$. Since the DFT length is $N = 64$, a spectral peak amplitude of

$32 = (1/2)64$ is what we expect, since

$$\mathrm{DFT}_k(\cos(\omega_x n)) \triangleq \sum_{n=0}^{N-1} \frac{e^{j\omega_x n} + e^{-j\omega_x n}}{2} e^{-j\omega_k n} = \sum_{n=0}^{N-1} \frac{e^{j0n}}{2} = \frac{N}{2}$$

when $\omega_k = \pm\omega_x$. This happens at bin numbers $k = (0.25/f_s)N = 16$ and $k = (0.75/f_s)N = 48$ for $N = 64$. However, recall that Matlab requires indexing from 1, so that these peaks will really show up at index 17 and 49 in the `magX` array.

The spectrum should be exactly zero at the other bin numbers. How accurately this happens can be seen by looking on a dB scale, as shown in Fig. 9.1c. We see that the spectral magnitude in the other bins is on the order of 300 dB lower, which is close enough to zero for audio work.

9.1.2 Example 2: FFT of a Not-So-Simple Sinusoid

Now let's increase the frequency in the above example by one-half of a bin:

```
% Example 2: Same as Example 1 but with a frequency between bins

f = 0.25 + 0.5/N;   % Move frequency off-center by half a bin

x = cos(2*pi*n*f*T);  % Signal to analyze
X = fft(x);           % Spectrum

% Plot time data
figure(2);
subplot(3,1,1);
plot(n,x,'*');
ni = [0:.1:N-1];      % Interpolated time axis
hold on;
plot(ni,cos(2*pi*ni*f*T),'-');
title('Sinusoid Sampled at NEAR 1/4 the Sampling Rate');
xlabel('Time (samples)'); ylabel('Amplitude');
text(-8,1,'a)'); hold off;

% Plot spectral magnitude
subplot(3,1,2);
magX = abs(X);
```

```
stem(fn,magX); grid;
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (Linear)');
text(-0.11,30,'b)');

% Same spectrum on a dB scale
subplot(3,1,3);
spec = 20*log10(magX);  % Spectral magnitude in dB
plot(fn,spec); grid
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)');
text(-0.11,40,'c)');
print -deps eps/example2.eps; hold off;
```

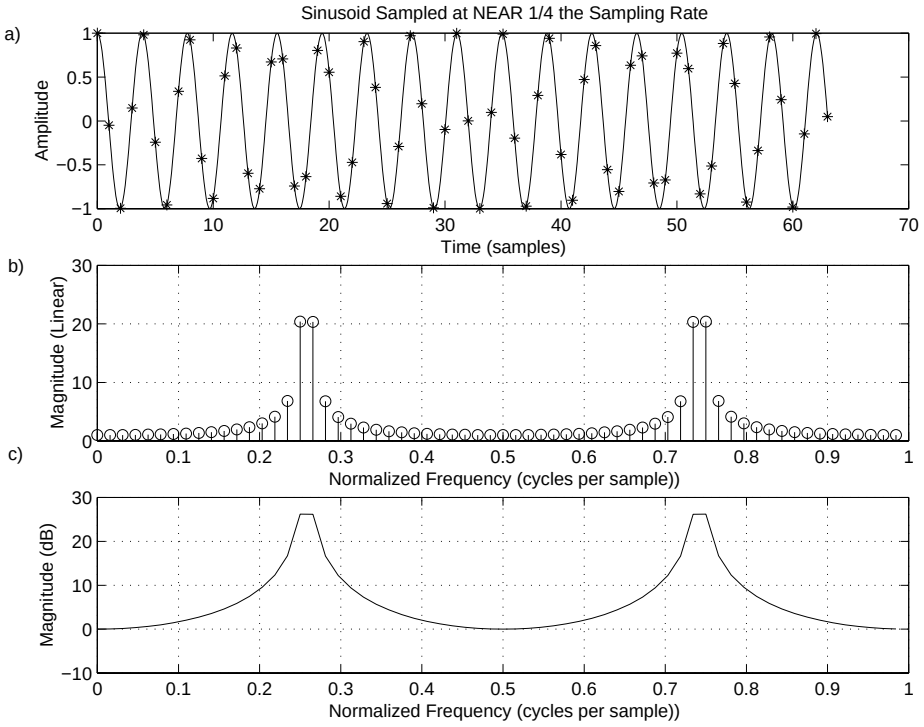

Figure 9.2: Sinusoid at Frequency $f = 0.25 + 0.5/N$. a) Time waveform. b) Magnitude spectrum. c) DB magnitude spectrum.

The resulting magnitude spectrum is shown in Fig. 9.2b and c. We see extensive "spectral leakage" into all the bins at this frequency.

To get an idea of where this spectral leakage is coming from, let's look at the periodic extension of the time waveform:

```
% Plot the periodic extension of the time-domain signal
plot([x,x]);
title('Time Waveform Repeated Once');
xlabel('Time (samples)');
ylabel('Amplitude');
print -deps eps/waveform2.eps;  % Figure 4
disp 'pausing for RETURN (check the plot). . .'; pause
```

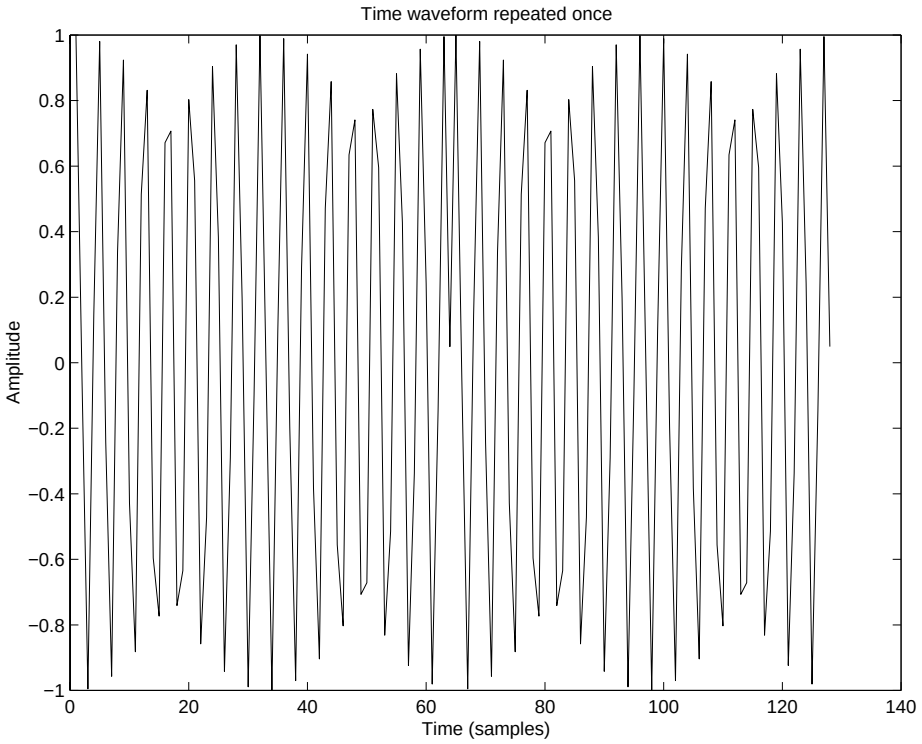

Figure 9.3: Time waveform repeated to show discontinuity introduced by periodic extension (see midpoint).

The result is shown in Fig. 9.3. Note the "glitch" in the middle where the signal begins its forced repetition.

9.1.3 Example 3: FFT of a Zero-Padded Sinusoid

Interestingly, looking back at Fig. 9.2c, we see there are no negative dB values. Could this be right? To *really* see the spectrum, let's use some *zero padding* in the time domain to yield ideal *interpolation* in the freqency domain:

```
% Example 3: Add zero padding
zpf = 8;                    % zero-padding factor
x = [cos(2*pi*n*f*T),zeros(1,(zpf-1)*N)];  % zero-padded FFT input data
X = fft(x);                  % Interpolated spectrum

% Plot time data
figure(4);
subplot(3,1,1);
plot(x);
title('Zero-Padded Sampled Sinusoid');
xlabel('Time (samples)'); ylabel('Amplitude');
text(-30,1,'a)'); hold off;

% Plot spectral magnitude
magX = abs(X);
nfft = zpf*N;
fni = [0:1.0/nfft:1-1.0/nfft];   % Normalized frequency axis
subplot(3,1,2);
plot(fni,magX,'-'); grid; % With interpolation, we can use solid lines '-'
% title('Interpolated Spectral Magnitude');
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (Linear)');
text(-.11,40,'b)');

% Same thing on a dB scale
spec = 20*log10(magX);  % Spectral magnitude in dB
spec = max(spec,-60*ones(1,length(spec))); % clip to -60 dB
subplot(3,1,3);
plot(fni,spec,'-'); grid; axis([0 1 -60 50]);
% title('Interpolated Spectral Magnitude (dB)');
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)');
text(-.11,50,'c)');
```

```
print -deps eps/example3.eps;
if dopause, disp 'pausing for RETURN (check the plot). . .'; pause; end
```

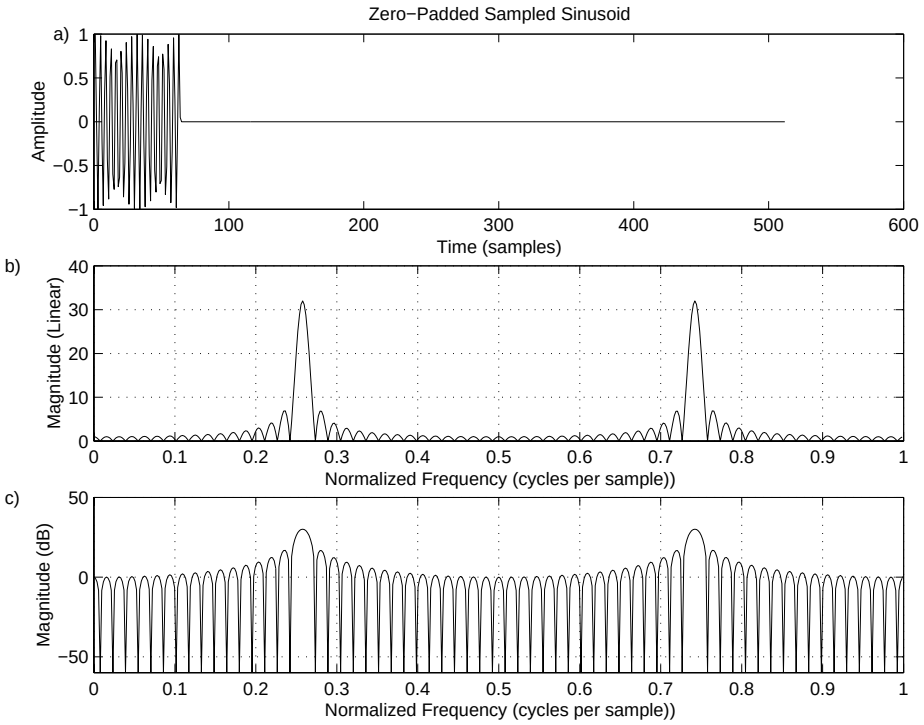

Figure 9.4: Zero-Padded Sinusoid at Frequency $f = 0.25 + 0.5/N$. a) Time waveform. b) Magnitude spectrum. c) DB magnitude spectrum.

With the zero padding, we see there's quite a bit going on. In fact, the spectrum has a regular *sidelobe* structure. On the dB scale in Fig. 9.4c, we now see that there are indeed negative dB values. This shows the importance of using zero padding to interpolate spectral displays so that the eye can "fill in" properly between the samples.

9.1.4 Example 4: Blackman Window

Finally, to finish off this sinusoid, let's look at the effect of using a *Blackman window* [22] (which has good though suboptimal characteristics for audio work). Figure 9.5a shows the Blackman window, Fig. 9.5b shows its magnitude spectrum on a dB scale, and Fig. 9.5c introduces the use of a more natural frequency axis which interprets the upper half of the bin numbers as negative frequencies. Here is the Matlab for it:

```
% Add a "Blackman window"
% w = blackman(N); % if you have the signal processing toolbox
w = .42-.5*cos(2*pi*(0:N-1)/(N-1))+.08*cos(4*pi*(0:N-1)/(N-1));
figure(5);
subplot(3,1,1); plot(w,'*'); title('The Blackman Window');
xlabel('Time (samples)'); ylabel('Amplitude');
text(-8,1,'a)');

% Also show the window transform:
xw = [w,zeros(1,(zpf-1)*N)]; % zero-padded window (col vector)
Xw = fft(xw);                % Blackman window transform
spec = 20*log10(abs(Xw));    % Spectral magnitude in dB
spec = spec - max(spec);     % Usually we normalize to 0 db max
spec = max(spec,-100*ones(1,nfft)); % clip to -100 dB
subplot(3,1,2); plot(fni,spec,'-'); axis([0,1,-100,10]); grid;
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)');
text(-.12,20,'b)');

% Replot interpreting upper bin numbers as negative frequencies:
nh = nfft/2;
specnf = [spec(nh+1:nfft),spec(1:nh)];  % see also Matlab's fftshift()
fninf = fni - 0.5;
subplot(3,1,3);
plot(fninf,specnf,'-'); axis([-0.5,0.5,-100,10]); grid;
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)');
text(-.6,20,'c)');
print -deps eps/blackman.eps;
```

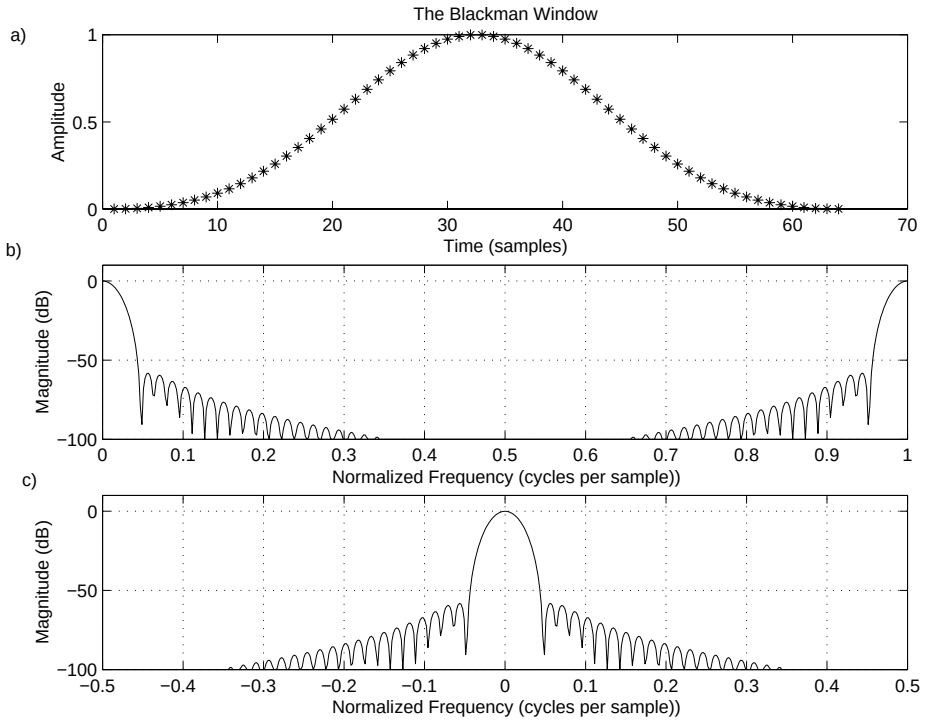

Figure 9.5: The Blackman Window. a) The window itself in the time domain. b) DB Magnitude spectrum of the Blackman window. c) Blackman-window DB magnitude spectrum plotted over frequencies $[-0.5, 0.5)$.

9.1.5 Example 5: Use of the Blackman Window

Now let's apply this window to the sinusoidal data:

```
% Use the Blackman window on the sinusoid data
xw = [w .* cos(2*pi*n*f*T),zeros(1,(zpf-1)*N)]; % windowed, zero-padded data
X = fft(xw);                                    % Smoothed, interpolated spectrum

% Plot time data
figure(6);
subplot(2,1,1);
plot(xw);
title('Windowed, Zero-Padded, Sampled Sinusoid');
xlabel('Time (samples)'); ylabel('Amplitude');
text(-50,1,'a)'); hold off;

% Plot spectral magnitude in the best way
spec = 10*log10(conj(X).*X);  % Spectral magnitude in dB
spec = max(spec,-60*ones(1,nfft)); % clip to -60 dB
subplot(2,1,2);
plot(fninf,fftshift(spec),'-'); axis([-0.5,0.5,-60,40]); grid;
title('Smoothed, Interpolated, Spectral Magnitude (dB)');
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)');
text(-.6,40,'b)');
print -deps eps/xw.eps;
```

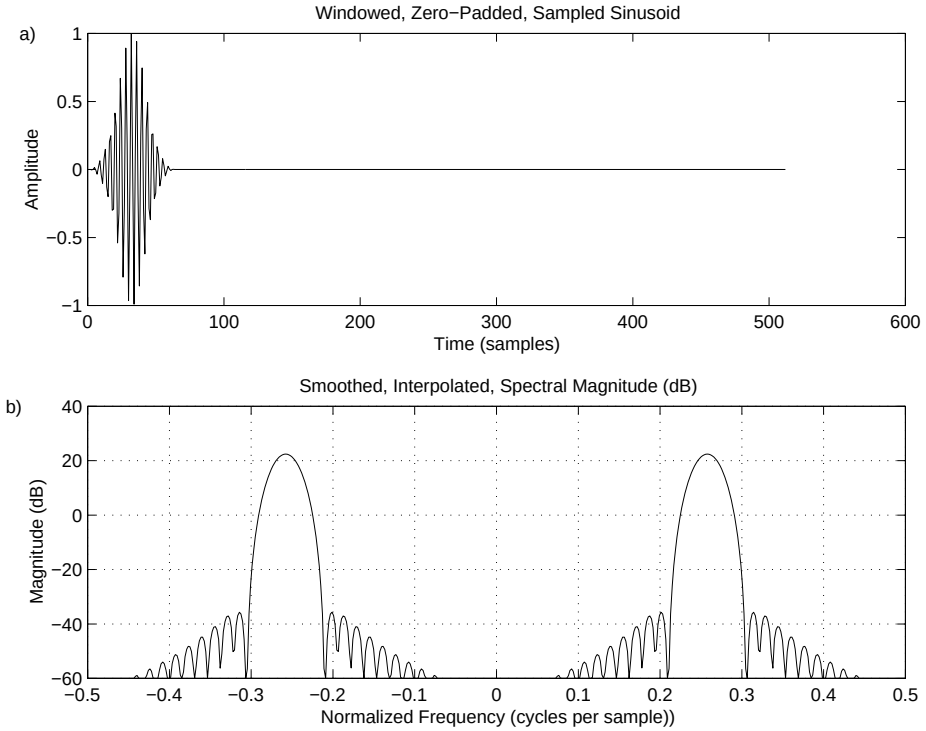

Figure 9.6: Effect of the Blackman window on the sinusoidal data segment.

9.1.6 Example 6: Hanning-Windowed Complex Sinusoid

In this example, we'll perform spectrum analysis on a complex sinusoid having only a single positive frequency. We'll use the Hanning window which does not have as much sidelobe suppression as the Blackman window, but its main lobe is narrower. Its sidelobes "roll off" very quickly versus frequency. Compare with the Blackman window results to see if you can see these differences.

```
% Example 5: Practical spectrum analysis of a sinusoidal signal

% Analysis parameters:
M = 31;         % Window length (we'll use a "Hanning window")
N = 64;         % FFT length (zero padding around a factor of 2)

% Signal parameters:
wxT = 2*pi/4;   % Sinusoid frequency in rad/sample (1/4 sampling rate)
A = 1;          % Sinusoid amplitude
phix = 0;       % Sinusoid phase

% Compute the signal x:
n = [0:N-1];    % time indices for sinusoid and FFT
x = A * exp(j*wxT*n+phix); % the complex sinusoid itself: [1,j,-1,-j,1,j,...]

% Compute Hanning window:
nm = [0:M-1];   % time indices for window computation
w = (1/M) * (cos((pi/M)*(nm-(M-1)/2))).^2;  % Hanning window = "raised cosine"
% (FIXME: normalizing constant above should be 2/M)

wzp = [w,zeros(1,N-M)]; % zero-pad out to the length of x
xw = x .* wzp;          % apply the window w to the signal x

% Display real part of windowed signal and the Hanning window:
plot(n,wzp,'-'); hold on;
plot(n,real(xw),'*');
title('Hanning Window and Windowed, Zero-Padded, Sinusoid (Real Part)');
xlabel('Time (samples)'); ylabel('Amplitude'); hold off;
disp 'pausing for RETURN (check the plot). . .'; pause
print -deps eps/hanning.eps;
```

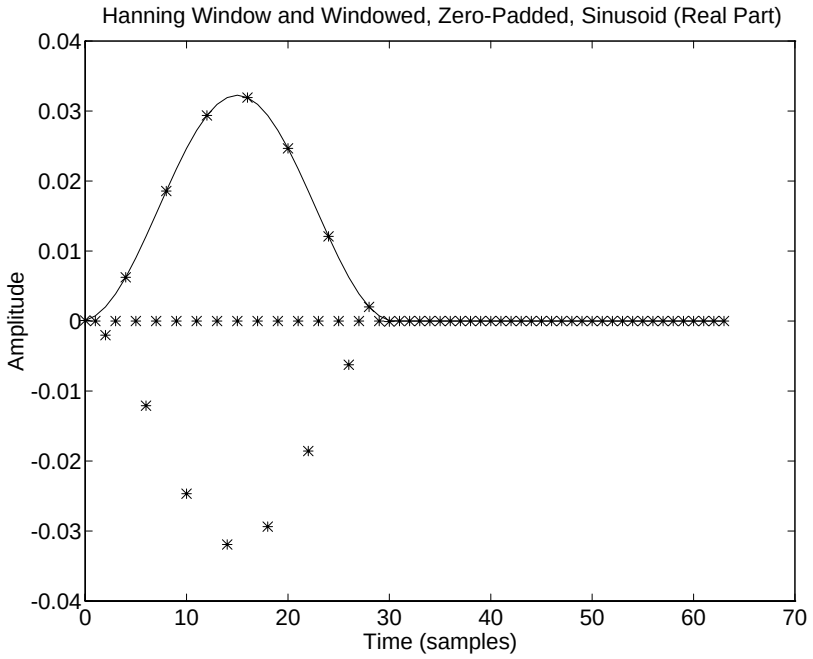

Figure 9.7: A length 31 Hanning Window ("Raised Cosine") and the windowed sinusoid created using it. Zero-padding is also shown. The sampled sinusoid is plotted with '*' using no connecting lines. You must now imagine the continuous sinusoid threading through the asterisks.

```
% Compute the spectrum and its various alternative forms
%
Xw = fft(xw);              % FFT of windowed data
fn = [0:1.0/N:1-1.0/N];    % Normalized frequency axis
spec = 20*log10(abs(Xw));  % Spectral magnitude in dB
% Since the zeros go to minus infinity, clip at -100 dB:
spec = max(spec,-100*ones(1,length(spec)));
phs = angle(Xw);           % Spectral phase in radians
phsu = unwrap(phs);        % Unwrapped spectral phase (using matlab function)
```

To help see the full spectrum, we'll also compute a heavily interpolated spectrum which we'll draw using solid lines. (The previously computed spectrum will be plotted using '*'.) Ideal spectral interpolation is done using zero-padding in the time domain:

```
Nzp = 16;                   % Zero-padding factor
Nfft = N*Nzp;               % Increased FFT size
xwi = [xw,zeros(1,Nfft-N)]; % New zero-padded FFT buffer
Xwi = fft(xwi);             % Take the FFT
fni = [0:1.0/Nfft:1.0-1.0/Nfft]; % Normalized frequency axis
speci = 20*log10(abs(Xwi));      % Interpolated spectral magnitude in dB
speci = max(speci,-100*ones(1,length(speci))); % clip at -100 dB
phsi = angle(Xwi);          % Phase
phsiu = unwrap(phsi);       % Unwrapped phase

% Plot spectral magnitude
%
plot(fn,abs(Xw),'*'); hold on; plot(fni,abs(Xwi));
title('Spectral Magnitude');
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Amplitude (Linear)');
disp 'pausing for RETURN (check the plot). . .'; pause
print -deps eps/specmag.eps; hold off;


% Same thing on a dB scale
plot(fn,spec,'*'); hold on; plot(fni,speci);
title('Spectral Magnitude (dB)');
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)');
```

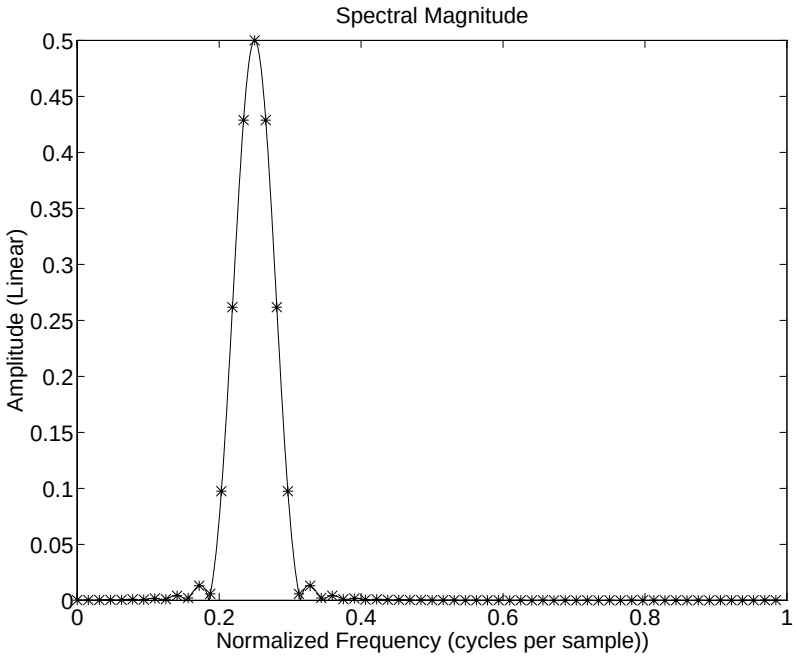

Figure 9.8: Spectral Magnitude, linear scale.

```
disp 'pausing for RETURN (check the plot). . .'; pause
print -deps eps/specmagdb.eps; hold off;
```

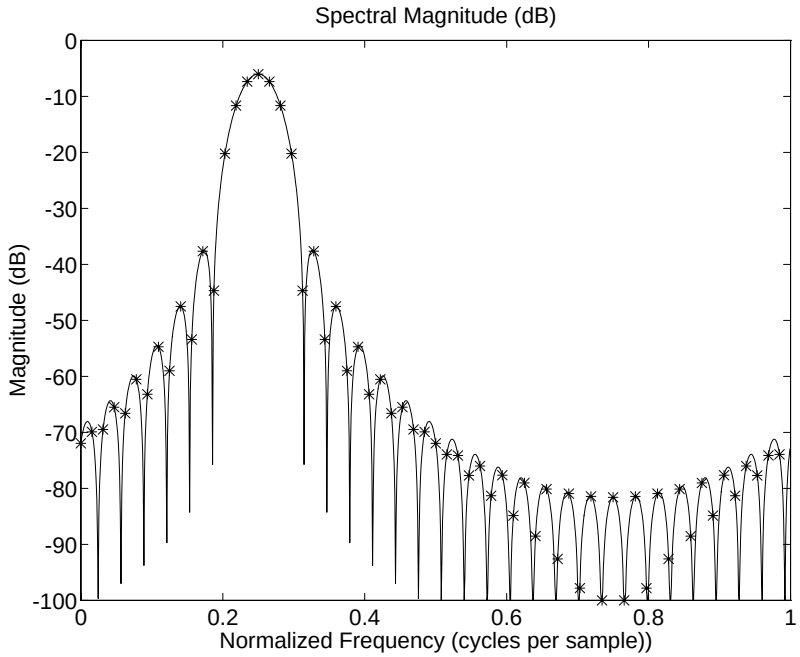

Figure 9.9: Spectral Magnitude, dB scale.

Note that there are no negative frequency components in Fig. 9.8 because we are analyzing a complex sinusoid $[1, j, -1, -j, 1, j, \ldots]$ which is a sampled complex sinusoid frequency $f_s/4$ only.

Notice how difficult it would be to correctly interpret the shape of the "sidelobes" without zero padding. The asterisks correspond to a zero-padding factor of 2, already twice as much as needed to preserve all spectral information faithfully, but it is clearly not sufficient to make the sidelobes clear in a spectral magnitude plot.

Spectral Phase

As for the phase of the spectrum, what do we expect? We have chosen the sinusoid phase to be zero. The window is symmetric about its middle. Therefore, we expect a linear phase term with slope -(M-1)/2 samples.

Also, the window transform has sidelobes which cause a phase of π radians to switch in and out. Thus, we expect to see samples of a straight line with slope -15 across the main lobe of the window transform, together with a switching offset by π in every other sidelobe away from the main lobe, starting with the immediately adjacent sidelobes.

In the plot, you can see the negatively sloped line across the main lobe of the window transform, but the sidelobes are hard to follow.

```
plot(fn,phs,'*'); hold on; plot(fni,phsi); grid;
title('Spectral Phase');
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Phase - Phi (Radians)');
disp 'pausing for RETURN (check the plot). . .'; pause
print -deps eps/specphase.eps; hold off;
```

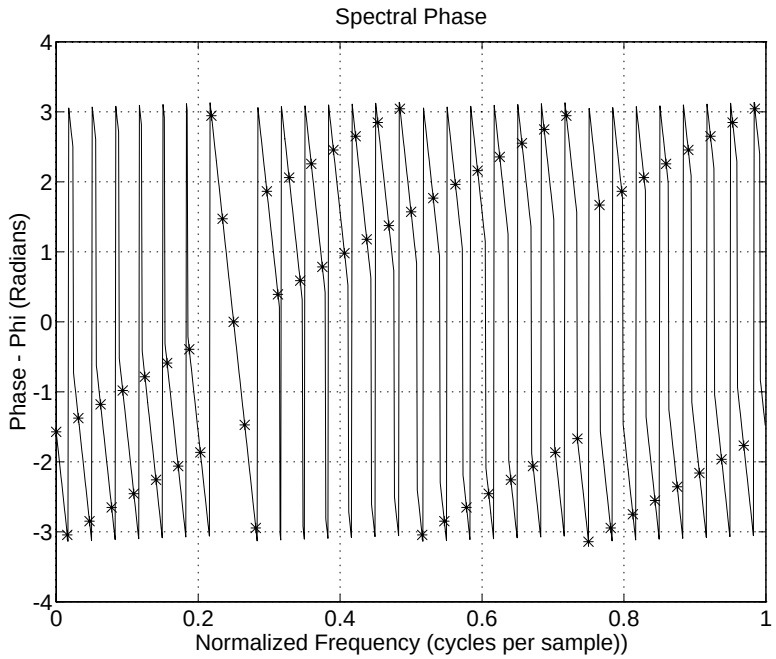

Figure 9.10: Spectral phase.

To convert the expected phase slope from -15 "radians per radian-frequency" to "radians per cycle-per-sample," we need to multiply by

"radians per cycle," or 2π. Thus, in Fig. 9.10, we expect a slope of -94.2 radians per unit normalized frequency, or -9.42 radians per 0.1 cycles-per-sample, and this looks about right, judging from the plot.

Appendix A

Matrices

A *matrix* is defined as a rectangular array of numbers, e.g.,

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

which is a 2×2 ("two by two") matrix. A general matrix may be $M \times N$, where M is the number of *rows*, and N is the number of *columns*. For example, the general 3×2 matrix is

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}$$

(Either square brackets or large parentheses may be used.) The (i,j)th element[1] of a matrix A may be denoted by $A[i,j]$ or $A(i,j)$. The rows and columns of matrices are normally numbered from 1 instead of from 0; thus, $1 \leq i \leq M$ and $1 \leq j \leq N$. When $N = M$, the matrix is said to be *square*.

The *transpose* of a real matrix $A \in \mathcal{R}^{M \times N}$ is denoted by A^{T} and is defined by

$$A^{\mathrm{T}}[i,j] \stackrel{\Delta}{=} A[j,i]$$

Note that while A is $M \times N$, its transpose is $N \times M$.

A *complex matrix* $A \in \mathcal{C}^{M \times N}$, is simply a matrix containing complex numbers. The *transpose* of a complex matrix is normally defined to

[1] We are now using j as an integer counter, not as $\sqrt{-1}$. This is standard notational practice.

include *conjugation*. The conjugating transpose operation is called the *Hermitian transpose*. To avoid confusion, in this tutorial, A^{T} and the word "transpose" will always denote transposition *without* conjugation, while conjugating transposition will be denoted by A^* and be called the "Hermitian transpose" or the "conjugate transpose." Thus,

$$A^*[i,j] \stackrel{\Delta}{=} \overline{A[j,i]}$$

Example: The transpose of the general 3×2 matrix is

$$\left[\begin{array}{cc} a & b \\ c & d \\ e & f \end{array}\right]^{\mathrm{T}} = \left[\begin{array}{ccc} a & c & e \\ b & d & f \end{array}\right]$$

while the conjugate transpose of the general 3×2 matrix is

$$\left[\begin{array}{cc} a & b \\ c & d \\ e & f \end{array}\right]^{*} = \left[\begin{array}{ccc} \overline{a} & \overline{c} & \overline{e} \\ \overline{b} & \overline{d} & \overline{f} \end{array}\right]$$

A *column-vector*

$$\underline{x} = \left[\begin{array}{c} x_0 \\ x_1 \end{array}\right]$$

is the special case of an $M \times 1$ matrix, and a *row-vector*

$$\underline{x}^{\mathrm{T}} = [x_0 \; x_1]$$

(as we have been using) is a $1 \times N$ matrix. In contexts where matrices are being used (only this section for this reader), it is best to define all vectors as *column vectors* and to indicate row vectors using the transpose notation, as was done in the equation above.

A.0.1 Matrix Multiplication

Let A^{T} be a general $M \times L$ matrix and let B denote a general $L \times N$ matrix. Denote the matrix product by $C = A^{\mathrm{T}}B$ or $C = A^{\mathrm{T}} \cdot B$. Then *matrix multiplication* is carried out by computing the *inner product* of every row of A^{T} with every column of B. Let the ith row of A^{T} be denoted by $\underline{a}_i^{\mathrm{T}}$, $i = 1, 2, \ldots, M$, and the jth column of B by $\underline{b}_j$, $j = 1, 2, \ldots, L$. Then the

matrix product $C = A^{\mathrm{T}}B$ is defined as

$$C = A^{\mathrm{T}}B = \left[\begin{array}{cccc} <\underline{a}_1^{\mathrm{T}},\underline{b}_1> & <\underline{a}_1^{\mathrm{T}},\underline{b}_2> & \cdots & <\underline{a}_1^{\mathrm{T}},\underline{b}_N> \\ <\underline{a}_2^{\mathrm{T}},\underline{b}_1> & <\underline{a}_2^{\mathrm{T}},\underline{b}_2> & \cdots & <\underline{a}_2^{\mathrm{T}},\underline{b}_N> \\ \vdots & \vdots & \cdots & \vdots \\ <\underline{a}_M^{\mathrm{T}},\underline{b}_1> & <\underline{a}_M^{\mathrm{T}},\underline{b}_2> & \cdots & <\underline{a}_M^{\mathrm{T}},\underline{b}_N> \end{array}\right]$$

This definition can be extended to *complex* matrices by using a definition of inner product which does not conjugate its second argument.[2]

Examples:

$$\left[\begin{array}{cc} a & b \\ c & d \\ e & f \end{array}\right] \cdot \left[\begin{array}{cc} \alpha & \beta \\ \gamma & \delta \end{array}\right] = \left[\begin{array}{cc} a\alpha+b\gamma & a\beta+b\delta \\ c\alpha+d\gamma & c\beta+d\delta \\ e\alpha+f\gamma & e\beta+f\delta \end{array}\right]$$

$$\left[\begin{array}{cc} \alpha & \beta \\ \gamma & \delta \end{array}\right] \cdot \left[\begin{array}{ccc} a & c & e \\ b & d & f \end{array}\right] = \left[\begin{array}{ccc} \alpha a+\beta b & \alpha c+\beta d & \alpha e+\beta f \\ \gamma a+\delta b & \gamma c+\delta d & \gamma e+\delta f \end{array}\right]$$

$$\left[\begin{array}{c} \alpha \\ \beta \end{array}\right] \cdot \left[\begin{array}{ccc} a & b & c \end{array}\right] = \left[\begin{array}{ccc} \alpha a & \alpha b & \alpha c \\ \beta a & \beta b & \beta c \end{array}\right]$$

$$\left[\begin{array}{ccc} a & b & c \end{array}\right] \cdot \left[\begin{array}{c} \alpha \\ \beta \\ \gamma \end{array}\right] = a\alpha + b\beta + c\gamma$$

An $M \times L$ matrix A can only be multiplied on the *right* by an $L \times N$ matrix, where N is any positive integer. An $L \times N$ matrix A can only be multiplied on the *left* by a $M \times L$ matrix, where M is any positive integer. Thus, the number of columns in the matrix on the left must equal the number of rows in the matrix on the right.

Matrix multiplication is *non-commutative*, in general. That is, normally $AB \neq BA$ even when both products are defined (such as when the matrices are square.)

The *transpose of a matrix product* is the product of the transposes in *reverse order*:

$$(AB)^{\mathrm{T}} = B^{\mathrm{T}}A^{\mathrm{T}}$$

[2]Alternatively, it can be extended to the complex case by writing $A^*B \triangleq [\ldots < b_j, a_i^* > \ldots]$, so that A^* includes a conjugation of the elements of A. This difficulty arises from the fact that matrix multiplication is really defined without consideration of conjugation or transposition at all, making it unwieldy to express in terms of inner products in the complex case, even though that is perhaps the most fundamental interpretation of a matrix multiply.

The *identity matrix* is denoted by I and is defined as

$$I \triangleq \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Identity matrices are always *square*. The $N \times N$ identity matrix I, sometimes denoted as I_N, satisfies $A \cdot I_N = A$ for every $M \times N$ matrix A. Similarly, $I_M \cdot A = A$, for every $M \times N$ matrix A.

As a special case, a matrix A^{T} times a vector $\underline{x}$ produces a new vector $\underline{y} = A^{\mathrm{T}}\underline{x}$ which consists of the inner product of every row of A^{T} with $\underline{x}$

$$A^{\mathrm{T}}\underline{x} = \begin{bmatrix} < \underline{a}_1^{\mathrm{T}}, \underline{x} > \\ < \underline{a}_2^{\mathrm{T}}, \underline{x} > \\ \vdots \\ < \underline{a}_M^{\mathrm{T}}, \underline{x} > \end{bmatrix}$$

A matrix A^{T} times a vector $\underline{x}$ defines a *linear transformation* of $\underline{x}$. In fact, every linear function of a vector $\underline{x}$ can be expressed as a matrix multiply. In particular, every linear *filtering* operation can be expressed as a matrix multiply applied to the input signal. As a special case, every linear, time-invariant (LTI) filtering operation can be expressed as a matrix multiply in which the matrix is *Toeplitz*, i.e., $A^{\mathrm{T}}[i,j] = A^{\mathrm{T}}[i-j]$ (constant along all *diagonals*).

As a further special case, a row vector on the left may be multiplied by a column vector on the right to form a *single inner product*:

$$\underline{y}^*\underline{x} = < \underline{x}, \underline{y} >$$

where the alternate transpose notation "$*$" is defined to include *complex conjugation* so that the above result holds also for complex vectors. Using this result, we may rewrite the general matrix multiply as

$$C = A^{\mathrm{T}}B = \begin{bmatrix} \underline{a}_1^{\mathrm{T}}\underline{b}_1 & \underline{a}_1^{\mathrm{T}}\underline{b}_2 & \cdots & \underline{a}_1^{\mathrm{T}}\underline{b}_N \\ \underline{a}_2^{\mathrm{T}}\underline{b}_1 & \underline{a}_2^{\mathrm{T}}\underline{b}_2 & \cdots & \underline{a}_2^{\mathrm{T}}\underline{b}_N \\ \vdots & \vdots & \vdots & \vdots \\ \underline{a}_M^{\mathrm{T}}\underline{b}_1 & \underline{a}_M^{\mathrm{T}}\underline{b}_2 & \cdots & \underline{a}_M^{\mathrm{T}}\underline{b}_N \end{bmatrix}$$

A.0.2 Solving Linear Equations Using Matrices

Consider the linear system of equations

$$\begin{aligned} ax_1 + bx_2 &= c \\ dx_1 + ex_2 &= f \end{aligned}$$

in matrix form:

$$\left[\begin{array}{cc} a & b \\ d & e \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \end{array}\right] = \left[\begin{array}{c} c \\ f \end{array}\right].$$

This can be written in higher level form as

$$A\underline{x} = \underline{b}$$

where A denotes the two-by-two matrix above, and $\underline{x}$ and $\underline{b}$ denotes the two-by-one vectors. The solution to this equation is then

$$\underline{x} = A^{-1}\underline{b} = \left[\begin{array}{cc} a & b \\ d & e \end{array}\right]^{-1} \left[\begin{array}{c} c \\ f \end{array}\right]$$

The general two-by-two matrix inverse is given by

$$\left[\begin{array}{cc} a & b \\ d & e \end{array}\right]^{-1} = \frac{1}{ae - bd} \left[\begin{array}{cc} e & -b \\ -d & a \end{array}\right]$$

and the inverse exists whenever $ae - bd$ (which is called the *determinant* of the matrix A) is nonzero. For larger matrices, numerical algorithms are used to invert matrices, such as used by Matlab based on LINPACK [23]. An initial introduction to matrices and linear algebra can be found in [16].

For a Stanford "courselet" on matrices, see
http://sll-1.stanford.edu/Courselets/Matrices/web/quiz/quizz.cgi/ask/quest.html.

Appendix B

Sampling Theory

A basic tutorial on sampling theory is presented. Aliasing due to sampling of continuous-time signals is characterized mathematically. Shannon's sampling theorem is proved. A pictorial representation of continuous-time signal reconstruction from discrete-time samples is given.

B.1 Introduction

Inside computers and modern "digital" synthesizers, (as well as music CDs), sound is *sampled* into a stream of *numbers*. Each *sample* can be thought of as a number which specifies the *position*[1] of a loudspeaker at a particular instant. When sound is sampled, we call it *digital audio*. The sampling rate used for CDs is 44,100 samples per second. That means

[1]More typically, each sample represents the instantaneous *velocity* of the speaker. Here's why: Most microphones are transducers from *acoustic pressure* to *electrical voltage*, and analog-to-digital converters (ADCs) produce numerical samples which are proportional to voltage. Thus, digital samples are normally proportional to *acoustic pressure deviation* (force per unit area on the microphone, with ambient air pressure subtracted out). When digital samples are converted to analog form by digital-to-analog conversion (DAC), each sample is converted to an electrical voltage which then drives a loudspeaker (in audio applications). Typical loudspeakers use a "voice-coil" to convert applied voltage to electromotive force on the speaker which applies pressure on the air via the speaker cone. Since the acoustic impedance of air is a real number, wave pressure is directly proportional wave *velocity*. Since the speaker must move in contact with the air during wave generation, we may conclude that digital signal samples correspond most closely to the *velocity* of the speaker, not its position. The situation is further complicated somewhat by the fact that speakers do not themselves have a real driving-point impedance. However, for an "ideal" microphone and speaker, we should get samples proportional to speaker velocity and hence to air pressure.

when you play a CD, the speakers in your stereo system are moved to a new position 44,100 times per second, or once every 23 microseconds. Controlling a speaker this fast enables it to generate any sound in the human hearing range because we cannot hear frequencies higher than around 20,000 cycles per second, and a sampling rate more than twice the highest frequency in the sound guarantees that exact reconstruction is possible from the samples.

B.1.1 Reconstruction from Samples—Pictorial Version

Figure B.1 shows how a sound is reconstructed from its samples. Each sample can be considered as specifying the *scaling* and *location* of a *sinc function*. The discrete-time signal being interpolated in the figure is $[\ldots,0,0,0,1,1,1,1,1,0,0,0,\ldots]$. The sinc functions are drawn with dashed lines, and they sum to produce the solid curve. Note the "Gibb's overshoot" near the corners of this continuous rectangular pulse due to band-limiting.

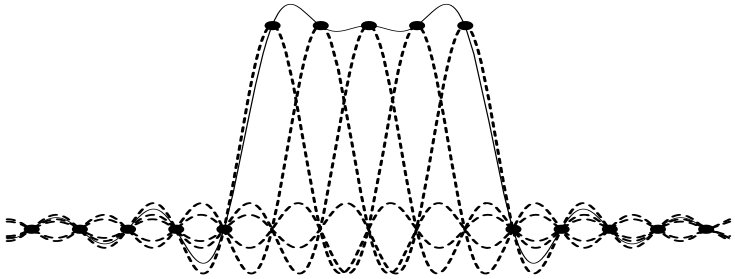

Figure B.1: How sinc functions sum up to create a continuous waveform from discrete-time samples.

Note how each sinc function passes through zero at every sample instant but the one it is centered on, where it passes through 1. An isolated sinc function is shown in Fig. B.2.

The sinc function is the famous "sine x over x" curve, defined by

$$\mathrm{sinc}(F_s t) \stackrel{\Delta}{=} \frac{\sin(\pi F_s t)}{\pi F_s t}.$$

where F_s denotes the sampling rate in samples-per-second (Hz), and t denotes time in seconds.

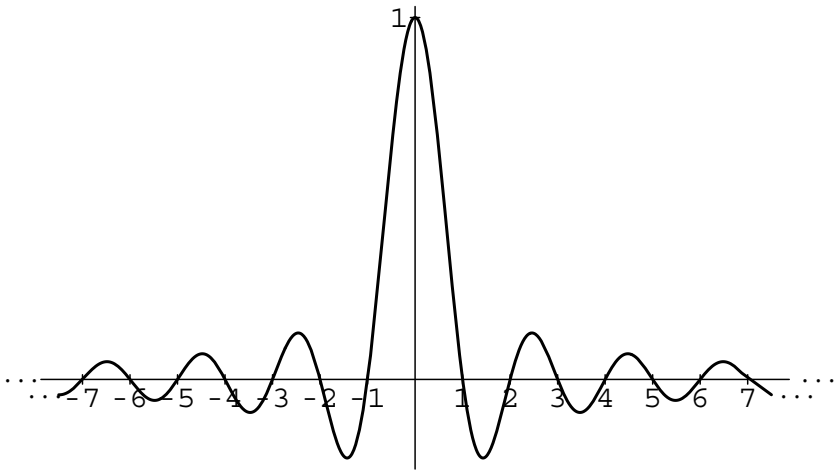

Figure B.2: The sinc function.

B.1.2 Reconstruction from Samples—The Math

Let $x_d(n) \triangleq x(nT_s)$ denote the nth sample of the original sound $x(t)$, where t is time in seconds. Thus, n ranges over the integers, and T_s is the *sampling period* in seconds. The *sampling rate* in Hertz (Hz) is just the reciprocal of the sampling period, *i.e.*,

$$F_s \triangleq \frac{1}{T_s}$$

To avoid losing any information as a result of sampling, we must assume $x(t)$ is *band-limited* to less than half the sampling rate. This means there can be no energy in $x(t)$ at frequency $F_s/2$ or above. We will prove this mathematically when we prove *Shannon's Sampling Theorem* in §B.3 below.

Let $X(\omega)$ denote the Fourier transform of $x(t)$, *i.e.*,

$$X(\omega) \triangleq \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt.$$

Then we can say x is *band-limited* to less than half the sampling rate if and only if $X(\omega) = 0$ for all $|\omega| \geq \pi F_s$. In this case, Shannon's sampling theorem gives us that $x(t)$ can be uniquely reconstructed from the samples

$x(nT_s)$ by summing up shifted, scaled, sinc functions:

$$\hat{x}(t) \stackrel{\Delta}{=} \sum_{n=-\infty}^{\infty} x(nT_s) h_s(t - nT_s) \equiv x(t)$$

where

$$h_s(t) \stackrel{\Delta}{=} \mathrm{sinc}(F_s t) \stackrel{\Delta}{=} \frac{\sin(\pi F_s t)}{\pi F_s t}$$

The sinc function is the impulse response of the *ideal lowpass filter*. This means its Fourier transform is a rectangular window in the frequency domain. The particular sinc function used here corresponds to the ideal lowpass filter which cuts off at half the sampling rate. In other words, it has a gain of 1 between frequencies 0 and $F_s/2$, and a gain of zero at all higher frequencies.

The reconstruction of a sound from its samples can thus be interpreted as follows: convert the sample stream into a *weighted impulse train*, and pass that signal through an ideal lowpass filter which cuts off at half the sampling rate. These are the fundamental steps of *digital to analog conversion* (DAC). In practice, neither the impulses nor the lowpass filter are ideal, but they are usually close enough to ideal that you cannot hear any difference. Practical lowpass-filter design is discussed in the context of *band-limited interpolation.*[2]

B.2 Aliasing of Sampled Signals

This section quantifies aliasing in the general case. This result is then used in the proof of Shannon's Sampling Theorem in the next section.

It is well known that when a continuous-time signal contains energy at a frequency higher than half the sampling rate $F_s/2$, then sampling at F_s samples per second causes that energy to *alias* to a lower frequency. If we write the original frequency as $f = F_s/2 + \epsilon$, then the new aliased frequency is $f_a = F_s/2 - \epsilon$, for $\epsilon \leq F_s/2$. This phenomenon is also called "folding", since f_a is a "mirror image" of f about $F_s/2$. As we will see, however, this is not a fundamental description of aliasing, as it only applies to real signals. For general (complex) signals, it is better to regard the aliasing due to sampling as a summation over all spectral "blocks" of width F_s.

[2] http://www-ccrma.stanford.edu/~jos/resample/

Theorem. (Continuous-Time Aliasing Theorem) Let $x(t)$ denote any continuous-time signal having a Fourier Transform (FT)

$$X(j\omega) \triangleq \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$$

Let

$$x_d(n) \triangleq x(nT_s), \quad n = \ldots, -2, -1, 0, 1, 2, \ldots,$$

denote the samples of $x(t)$ at uniform intervals of T_s seconds, and denote its Discrete-Time Fourier Transform (DTFT) by

$$X_d(e^{j\theta}) \triangleq \sum_{n=-\infty}^{\infty} x_d(n)e^{-j\theta n}$$

Then the spectrum X_d of the sampled signal x_d is related to the spectrum X of the original continuous-time signal x by

$$X_d(e^{j\theta}) = \frac{1}{T_s}\sum_{m=-\infty}^{\infty} X\left[j\left(\frac{\theta}{T_s} + m\frac{2\pi}{T_s}\right)\right].$$

The terms in the above sum for $m \neq 0$ are called *aliasing terms*. They are said to *alias* into the *base band* $[-\pi/T_s, \pi/T_s]$. Note that the summation of a spectrum with aliasing components involves addition of complex numbers; therefore, aliasing components can be removed only if both their *amplitude and phase* are known.

Proof. Writing $x(t)$ as an inverse FT gives

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(j\omega)e^{j\omega t}d\omega$$

Writing $x_d(n)$ as an inverse DTFT gives

$$x_d(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X_d(e^{j\theta})x(t)e^{j\theta t}d\theta$$

where $\theta \triangleq 2\pi\omega_d T_s$ denotes the normalized discrete-time frequency variable.

The inverse FT can be broken up into a sum of finite integrals, each of length $\Omega_s \triangleq 2\pi F_s = 2\pi/T_s$, as follows:

$$\begin{aligned}
x(t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} X(j\omega)e^{j\omega t}d\omega \\
&= \frac{1}{2\pi}\sum_{m=-\infty}^{\infty}\int_{(2m-1)\pi/T_s)}^{(2m+1)\pi/T_s} X(j\omega)e^{j\omega t}d\omega \\
&\overset{\omega \leftarrow m\Omega_s}{=} \frac{1}{2\pi}\sum_{m=-\infty}^{\infty}\int_{-\Omega_s/2}^{\Omega_s/2} X\left(j\omega + jm\Omega_s\right)e^{j\omega t}e^{j\Omega_s m t}d\omega \\
&= \frac{1}{2\pi}\int_{-\Omega_s/2}^{\Omega_s/2} e^{j\omega t}\sum_{m=-\infty}^{\infty} X\left(j\omega + jm\Omega_s\right)e^{j\Omega_s m t}d\omega
\end{aligned}$$

Let us now sample this representation for $x(t)$ at $t = nT_s$ to obtain

$$\begin{aligned}
x_d(n) \triangleq x(nT_s) &= \frac{1}{2\pi}\int_{-\Omega_s/2}^{\Omega_s/2} e^{j\omega nT_s}\sum_{m=-\infty}^{\infty} X\left(j\omega + jm\Omega_s\right)e^{j\Omega_s mnT_s}d\omega \\
&= \frac{1}{2\pi}\int_{-\Omega_s/2}^{\Omega_s/2} e^{j\omega nT_s}\sum_{m=-\infty}^{\infty} X\left(j\omega + jm\Omega_s\right)e^{j(2\pi/T_s)mnT_s}d\omega \\
&= \frac{1}{2\pi}\int_{-\Omega_s/2}^{\Omega_s/2} e^{j\omega nT_s}\sum_{m=-\infty}^{\infty} X\left(j\omega + jm\Omega_s\right)d\omega
\end{aligned}$$

since n and m are integers. Normalizing frequency as $\theta' = \omega T_s$ yields

$$x_d(n) = \frac{1}{2\pi}\int_{-\pi}\pi e^{j\theta' n}\frac{1}{T_s}\sum_{m=-\infty}^{\infty} X\left[j\left(\frac{\theta'}{T_s} + m\frac{2\pi}{T_s}\right)\right]d\theta'.$$

Since this is formally the inverse DTFT of $X_d(e^{j\theta'})$ written in terms of $X(j\theta'/T_s)$, the result follows. □

B.3 Shannon's Sampling Theorem

Theorem. Let $x(t)$ denote any continuous-time signal having a *continuous* Fourier transform

$$X(j\omega) \triangleq \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$$

Let

$$x_d(n) \triangleq x(nT_s), \quad n = \ldots, -2, -1, 0, 1, 2, \ldots,$$

denote the samples of $x(t)$ at uniform intervals of T_s seconds. Then $x(t)$ can be exactly reconstructed from its samples $x_d(n)$ if and only if $X(j\omega) = 0$ for all $|\omega| \geq \pi/T_s$.[3]

Proof. From the Continuous-Time Aliasing Theorem of §B.2, we have that the discrete-time spectrum $X_d(e^{j\theta})$ can be written in terms of the continuous-time spectrum $X(j\omega)$ as

$$X_d(e^{j\omega_d T_s}) = \frac{1}{T_s}\sum_{m=-\infty}^{\infty} X[j(\omega_d + m\Omega_s)]$$

where $\omega_d \triangleq \theta/T_s$ is the "digital frequency" variable. If $X(j\omega) = 0$ for all $|\omega| \geq \Omega_s/2$, then the above infinite sum reduces to one term, the $m = 0$ term, and we have

$$X_d(e^{j\omega_d T_s}) = \frac{1}{T_s}X(j\omega_d), \quad \omega_d \in [-\pi/T_s, \pi/T_s]$$

At this point, we can see that the spectrum of the sampled signal $x(nT_s)$ coincides with the spectrum of the continuous-time signal $x(t)$. In other words, the DTFT of $x(nT_s)$ is equal to the FT of $x(t)$ between plus and minus half the sampling rate, and the FT is zero outside that range. This makes it clear that spectral information is preserved, so it should now be possible to go from the samples back to the continuous waveform without error.

[3]Mathematically, $X(j\omega)$ can be allowed to be nonzero over points $|\omega| \geq \pi/T_s$ provided that the set of all such points have *measure zero* in the sense of Lebesgue integration. However, such distinctions do not arise for practical signals which are always finite in extent and which therefore have continuous Fourier transforms. This is why we specialize Shannon's Sampling Theorem to the case of continuous-spectrum signals.

To reconstruct $x(t)$ from its samples $x(nT_s)$, we may simply take the inverse Fourier transform of the zero-extended DTFT, i.e.,

$$\begin{aligned} x(t) &= \mathrm{IFT}_t(X) \triangleq \frac{1}{2\pi}\int_{-\infty}^{\infty} X(j\omega)e^{j\omega t}d\omega = \frac{1}{2\pi}\int_{-\Omega_s/2}^{\Omega_s/2} X(j\omega)e^{j\omega t}d\omega \\ &= \frac{1}{2\pi}\int_{-\Omega_s/2}^{\Omega_s/2} X_d(e^{j\theta})e^{j\omega t}d\omega \triangleq \mathrm{IDTFT}_t(X_d) \end{aligned}$$

By expanding $X_d(e^{j\theta})$ as the DTFT of the samples $x(n)$, the formula for reconstructing $x(t)$ as a superposition of sinc functions weighted by the samples, depicted in Fig. B.1, is obtained:

$$\begin{aligned} x(t) &= \mathrm{IDTFT}_t(X_d) \\ &\triangleq \frac{1}{2\pi}\int_{-\pi}^{\pi} X_d(e^{j\theta})e^{j\omega t}d\omega \\ &= \frac{T_s}{2\pi}\int_{-\pi/T_s}^{\pi/T_s} X_d(e^{j\omega_d T_s})e^{j\omega_d t}d\omega_d \\ &\triangleq \frac{T_s}{2\pi}\int_{-\pi/T_s}^{\pi/T_s}\left[\sum_{n=-\infty}^{\infty} x(nT_s)e^{-j\omega_d nT_s}\right]e^{j\omega_d t}d\omega_d \\ &= \sum_{n=-\infty}^{\infty} x(nT_s)\underbrace{\frac{T_s}{2\pi}\int_{-\pi/T_s}^{\pi/T_s} e^{j\omega_d(t-nT_s)}d\omega_d}_{\triangleq h(t-nT_s)} \\ &\triangleq \sum_{n=-\infty}^{\infty} x(nT_s)h(t-nT_s) \\ &\triangleq (x*h)(t) \end{aligned}$$

where we defined

$$\begin{aligned} h(t-nT_s) &\triangleq \frac{T_s}{2\pi}\int_{-\pi/T_s}^{\pi/T_s} e^{j\omega_d(t-nT_s)}d\omega_d \\ &= \frac{T_s}{2\pi}\frac{2}{2j(t-nT_s)}\left[e^{j\pi\frac{t-n}{Ts}/T_s} - e^{-j\pi\frac{t-nT_s}{T_s}}\right] \\ &= \frac{sin\left[\pi\left(\frac{t}{T_s}-n\right)\right]}{\pi\left(\frac{t}{T_s}-n\right)} \\ &\triangleq \mathrm{sinc}(t-nT_s) \end{aligned}$$

I.e.,

$$h(t) = \mathrm{sinc}(t) \triangleq \frac{\sin(\pi t)}{\pi t}$$

The "sinc function" is defined with π in its argument so that it has zero crossings on the integers, and its peak magnitude is 1. Figure B.2 illustrates the appearance of the sinc function.

We have shown that when $x(t)$ is band-limited to less than half the sampling rate, the IFT of the zero-extended DTFT of its samples $x(nT_s)$ gives back the original continuous-time signal $x(t)$.

Conversely, if $x(t)$ can be reconstructed from its samples $x_d(n) \triangleq x(nT_s)$, it must be true that $x(t)$ is band-limited to $(-F_s/2, F_s/2)$, since a sampled signal only supports frequencies up to $F_s/2$ (see Appendix B.4). This completes the proof of Shannon's Sampling Theorem. □

A "one-line summary" of Shannon's sampling theorem is as follows:

$$x(t) = \mathrm{IFT}_t\left\{\textsc{ZeroPad}_\infty\left\{\mathrm{DTFT}\{x_d\}\right\}\right\}$$

That is, the domain of the Discrete-Time Fourier Transform of the samples is extended to plus and minus infinity by zero ("zero padded"), and the inverse Fourier transform of that gives the original signal. The Continuous-Time Aliasing Theorem provides that the zero-padded $\mathrm{DTFT}\{x_d\}$ and the original signal spectrum $\mathrm{FT}\{x\}$ are identical, as needed.

Shannon's sampling theorem is easier to show when applied to discrete-time sampling-rate conversion, i.e., when simple decimation of a discrete time signal is being used to reduce the sampling rate by an integer factor. In analogy with the Continuous-Time Aliasing Theorem of §B.2, the Decimation Theorem states that downsampling a digital signal by an integer factor L produces a digital signal whose spectrum can be calculated by partitioning the original spectrum into L equal blocks and then summing (aliasing) those blocks. If only one of the blocks is nonzero, then the original signal at the higher sampling rate is exactly recoverable.

B.4 Another Path to Sampling Theory

Consider $z_0 \in \mathcal{C}$, with $|z_0| = 1$. Then we can write z_0 in polar form as

$$z_0 \triangleq e^{j\theta_0} \triangleq e^{j\omega_0 T_s},$$

where θ_0, ω_0, and T_s are real numbers.

Forming a geometric sequence based on z_0 yields the sequence

$$x(t_n) \stackrel{\Delta}{=} z_0^n = e^{j\theta_0 n} = e^{j\omega_0 t_n}$$

where $t_n \stackrel{\Delta}{=} nT_s$. Thus, successive integer powers of z_0 produce a *sampled complex sinusoid* with unit amplitude, and zero phase. Defining the *sampling interval* as T_s in seconds provides that ω_0 is the *radian frequency* in radians per second.

B.4.1 What frequencies are representable by a geometric sequence?

A natural question to investigate is what frequencies ω_0 are possible. The angular step of the point z_0^n along the unit circle in the complex plane is $\theta_0 = \omega_0 T_s$. Since $e^{j(\theta_0 n + 2\pi)} = e^{j\theta_0 n}$, an angular step $\theta_0 > 2\pi$ is indistinguishable from the angular step $\theta_0 - 2\pi$. Therefore, we must restrict the angular step θ_0 to a length 2π range in order to avoid ambiguity.

Recall that we need support for both positive and negative frequencies since equal magnitudes of each must be summed to produce real sinusoids, as indicated by the identities

$$\begin{aligned}\cos(\omega_0 t_n) &= \frac{1}{2}e^{j\omega_0 t_n} + \frac{1}{2}e^{-j\omega_0 t_n}\\ \sin(\omega_0 t_n) &= -\frac{j}{2}e^{j\omega_0 t_n} + \frac{j}{2}e^{-j\omega_0 t_n}.\end{aligned}$$

The length 2π range which is symmetric about zero is

$$\theta_0 \in [-\pi, \pi],$$

which, since $\theta_0 = \omega_0 T_s = 2\pi f_0 T_s$, corresponds to the frequency range

$$\begin{aligned}\omega_0 &\in [-\pi/T_s, \pi/T_s]\\ f_0 &\in [-F_s/2, F_s/2]\end{aligned}$$

However, there is a problem with the point at $f_0 = \pm F_s/2$: Both $+F_s/2$ and $-F_s/2$ correspond to the same point $z_0 = -1$ in the z-plane. Technically, this can be viewed as *aliasing* of the frequency $-F_s/2$ on top of $F_s/2$, or vice versa. The practical impact is that it is not possible in general to reconstruct a sinusoid from its samples at this frequency. For an obvious example, consider the sinusoid at half the sampling-rate sampled

on its zero-crossings: $\sin(\omega_0 t_n) = \sin(\pi n) \equiv 0$. We cannot be expected to reconstruct a nonzero signal from a sequence of zeros! For the signal $\cos(\omega_0 t_n) = \cos(\pi n) = (-1)^n$, on the other hand, we sample the positive and negative peaks, and everything looks looks fine. In general, we either do not know the amplitude, or we do not know phase of a sinusoid sampled at exactly twice its frequency, and if we hit the zero crossings, we lose it altogether.

In view of the foregoing, we may define the valid range of "digital frequencies" to be

$$\begin{aligned} \theta_0 &\in [-\pi, \pi) \\ \omega_0 &\in [-\pi/T_s, \pi/T_s) \\ f_0 &\in [-F_s/2, F_s/2) \end{aligned}$$

While you might have expected the open interval $(-\pi, \pi)$, we are free to give the point $z_0 = -1$ either the largest positive or largest negative representable frequency. Here, we chose the largest negative frequency since it corresponds to the assignment of numbers in two's complement arithmetic, which is often used to implement phase registers in sinusoidal oscillators. Since there is no corresponding positive-frequency component, samples at $F_s/2$ must be interpreted as samples of clockwise circular motion around the unit circle at two points. Such signals are any alternating sequence of the form $c(-1)^n$, where c can be be complex. The amplitude at $-F_s/2$ is then defined as $|c|$, and the phase is $\angle c$.

We have seen that uniformly spaced samples can represent frequencies f_0 only in the range $[-F_s/2, F_s/2)$, where F_s denotes the sampling rate. Frequencies outside this range yield sampled sinusoids indistinguishable from frequencies inside the range.

Suppose we henceforth agree to sample at *higher* than twice the highest frequency in our continuous-time signal. This is normally ensured in practice by lowpass-filtering the input signal to remove all signal energy at $F_s/2$ and above. Such a filter is called an *anti-aliasing filter*, and it is a standard first stage in an *Analog-to-Digital (A/D) Converter (ADC)*. Nowadays, ADCs are normally implemented within a single integrated circuit chip, such as a CODEC (for "coder/decoder") or "multimedia chip".

B.4.2 Recovering a Continuous Signal from its Samples

Given samples of a properly band-limited signal, how do we reconstruct the original continuous waveform? I.e., given $x(t_n), n = 0, 1, 2, \ldots, N-1$, how do we compute $x(t)$ for any value of t?

One reasonable definition for $x(t)$ can be based on the DFT of $x(n)$:

$$X(e^{j\omega_k}) \triangleq \sum_{n=0}^{N-1} x(t_n)e^{-j\omega_k t_n}, \quad k = 0, 1, 2, \ldots, N-1$$

Since $X(e^{j\omega_k})$ gives the magnitude and phase of the sinusoidal component at frequency ω_k, we simply construct $x(t)$ as the sum of its constituent sinusoids, using continuous-time versions of the sinusoids:

$$x(t) \triangleq \sum_{k=0}^{N-1} X(e^{j\omega_k})e^{j\omega_k t}$$

This method is based on the fact that we know how to reconstruct sampled complex sinusoids exactly, since we have a "closed form" formula for any sinusoid. This method makes perfectly fine sense, but note that this definition of $x(t)$ is *periodic* with period NT_s seconds. This happens because each of the sinusoids $e^{j\omega_k t}$ repeats after NT_s seconds. This is known as the *periodic extension* property of the DFT, and it results from the fact that we had only N samples to begin with, so some assumption must be made outside that interval. We see that the "automatic" assumption built into the math is periodic extension. However, in practice, it is far more common to want to assume zeros outside the range of samples:

$$x(t) \triangleq \begin{cases} \sum_{k=0}^{N-1} X(e^{j\omega_k})e^{j\omega_k t}, & 0 \le t \le (N-1)T_s \\ 0, & \text{otherwise} \end{cases}$$

Note that the nonzero time interval can be chosen as any length NT_s interval. Often the best choice is $[-T_s(N-1)/2, T_s(N-1)/2 - 1]$, which allows for both positive and negative times. ("Zero-phase filtering" must be implemented using this convention, for example.)

"Chopping" the sum of the N "DFT sinusoids" in this way to zero outside an N-sample range works fine if the original signal $x(t_n)$ starts out and finishes with zeros. Otherwise, however, the truncation will cause errors at the edges, as can be understood from Fig. B.1.

Does it Work?

It is straightforward to show that the "additive synthesis" reconstruction method of the previous section actually works exactly (in the periodic case) in the following sense:

- The reconstructed signal $x(t)$ is *band-limited* to $[-\pi, \pi)$, i.e., its *Fourier transform* $X(\omega)$ is zero for all $|\omega| > \pi/T_s$. (This is not quite true in the truncated case.)
- The reconstructed signal $x(t)$ passes through the samples $x(t_n)$ exactly. (This is true in both cases.)

Is this enough? Are we done? Well, not quite. We know by construction that $x(t)$ is *a* band-limited interpolation of $x(t_n)$. But are band-limited interpolations unique? If so, then this must be it, but are they unique? The answer turns out to be yes, based on Shannon's Sampling Theorem. The uniqueness follows from the uniqueness of the inverse Fourier transform. We still have two different cases, however, depending on whether we assume periodic extension or zero extension beyond the range $n \in [0, N-1]$. In the periodic case, we have found the answer; in the zero-extended case, we need to use the sum-of-sincs construction provided in the proof of Shannon's sampling theorem.

Why do the DFT sinusoids suffice for interpolation in the periodic case and not in the zero-extended case? In the periodic case, the spectrum consists of the DFT frequencies and *nothing else*, so additive synthesis using DFT sinusoids works perfectly. A sum of N DFT sinusoids can *only* create a periodic signal (since each of the sinusoids repeats after N samples). Truncating such a sum in time results in *all* frequencies being present to some extent (save isolated points) from $\omega = 0$ to $\omega = \infty$. Therefore, the truncated result is not band-limited, so it must be wrong.

It is a well known Fourier fact that *no function can be both time-limited and band-limited.* Therefore, any truly band-limited interpolation must be a function which has infinite duration, such as the sinc function $\mathrm{sinc}(F_s t)$ used in bandlimited interpolation by a sum of sincs. Note that such a sum of sincs does pass through zero at all sample times in the "zero extension" region.

...

Bibliography

[1] R. V. Churchill, *Complex Variables and Applications*, McGraw-Hill, New York, 1960.

[2] W. R. LePage, *Complex Variables and the Laplace Transform for Engineers*, Dover, New York, 1961.

[3] J. Gullberg, *Mathematics From the Birth of Numbers*, Norton and Co., New York, 1997, [Qa21.G78 1996] ISBN 0-393-04002-X.

[4] J. O. Smith, "Introduction to digital filter theory", in *Digital Audio Signal Processing: An Anthology*, J. Strawn, Ed. William Kaufmann, Inc., Los Altos, California, 1985, (out of print). Updated version available online at http://www-ccrma.stanford.edu/~jos/filters/. Original version available as Stanford University Department of Music Technical Report STAN–M–20, April 1985. A shortened version appears in [25].

[5] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.

[6] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.

[7] A. D. Pierce, *Acoustics*, American Institute of Physics, for the Acoustical Society of America, (516)349-7800 x 481, 1989.

[8] S. S. Stevens and H. Davis, *Hearing: It's Psychology and Physiology*, American Institute of Physics, for the Acoustical Society of America, (516)349-7800 x 481, 1983, Copy of original 1938 edition.

[9] J. Dattorro, "The implementation of recursive digital filters for high-fidelity audio", *Journal of the Audio Engineering Society*, vol. 36,

pp. 851–878, Nov. 1988, Comments, ibid. (Letters to the Editor), vol. 37, p. 486 (1989 June); Comments, ibid. (Letters to the Editor), vol. 38, pp. 149-151 (1990 Mar.).

[10] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, Englewood Cliffs, NJ, 1978.

[11] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, New York, 1965.

[12] L. L. Sharf, *Statistical Signal Processing, Detection, Estimation, and Time Series Analysis*, Addison-Wesley, Reading MA, 1991.

[13] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation*, Prentice-Hall, Inc., Englewood Cliffs, NJ, April 2000.

[14] K. Steiglitz, *A Digital Signal Processing Primer with Applications to Audio and Computer Music*, Addison-Wesley, Reading MA, 1996.

[15] C. Dodge and T. A. Jerse, *Computer Music*, Schirmer, New York, 1985.

[16] B. Noble, *Applied Linear Algebra*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1969.

[17] M. Abramowitz and I. A. Stegun, Eds., *Handbook of Mathematical Functions*, Dover, New York, 1965.

[18] R. Bracewell, *The Fourier Transform and its Applications*, McGraw-Hill, New York, 1965.

[19] E. O. Brigham, *The Fast Fourier Transform*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974.

[20] S. M. Kay, *Modern Spectral Estimation*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1988.

[21] L. Ljung and T. L. Soderstrom, *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA, 1983.

[22] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform", *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, Jan 1978.

[23] G. H. Golub and C. F. Van Loan, *Matrix Computations, 2nd Edition*, The Johns Hopkins University Press, Baltimore, 1989.

[24] J. H. McClellan, R. W. Schafer, and M. A. Yoder, *DSP First: A Multimedia Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1998, Tk5102.M388.

[25] C. Roads, Ed., *The Music Machine*, MIT Press, Cambridge, MA, 1989.

[26] H. L. F. von Helmholtz, *Die Lehre von den Tonempfindungen als Physiologische Grundlage für die Theorie der Musik*, F. Vieweg und Sohn, Braunschweig, 1863.

[27] C. Roads, *The Computer Music Tutorial*, MIT Press, Cambridge, MA, 1996.

[28] N. H. Fletcher and T. D. Rossing, *The Physics of Musical Instruments, Second Edition*, Springer Verlag, New York, 1998, 485 illustrations, hardcover, 69.95 USD direct from Springer.

Index

20 dB boost, **45**
3 dB boost, **46**

absolute value of a complex number, **14**
aliased sinc function, **134**
Aliasing, **160**
aliasing operator, **161**
amplitude response, **181**
anti-aliasing lowpass filter, **163**
antilog, **41**
antilogarithm, **41**
antisymmetric functions, 163
argument of a complex number, **14**
average power, **66**

bandlimited interpolation in the frequency domain, **156**
bandlimited interpolation in the time domain, **156**
base, **41**
base 10 logarithm, 42
base 2 logarithm, 42
bel, **44**
bin (discrete Fourier transform), **136**
bin number, **136**
binary, 53
binary digits, **53**
bits, 53

carrier, **87**
Cartesian coordinates, **14**
characteristic of the logarithm, **42**
circular cross-correlation, **182**
CODEC, **64**
coefficient of projection, **132**
coherence function, **187**
column-vector, **212**
comb-filter, **75**
common logarithm, **41**
commutativity of convolution, **151**
companding, **52**, **64**
complex amplitude, **88**
complex conjugate, **14**
complex matrix, **211**
complex matrix transpose, 211
complex multiplication, **12**
complex number, **12**
complex numbers, **10**
    argument or angle or phase, **14**
    modulus or magnitude or radius or absolute value or norm, **14**
complex plane, **13**
conjugation in the frequency domain corresponds to reversal in the time domain, **167**
convolution, **151**
convolution in the frequency domain, **173**

convolution representation, **180**
correlation operator, **154**
cross-correlation, circular, **182**
cross-covariance, **187**
cross-spectral density, **183**
cross-talk, **134**
cyclic convolution, 151

dB per decade, **45**
dB per octave, **46**
dB scale, **44**
De Moivre's theorem, **17**
decibel, **44**
decimal numbers, **54**
DFT as a digital filter, 133
DFT matrix, **139**
digit, 54
digital filter, **180**
Discrete Fourier Transform, **145**
Discrete Fourier Transform (DFT), **1**
dynamic range, **52**
dynamic range of magnetic tape, 52

energy, **44**
Euler's Formula, **15**
Euler's Theorem, **27**
even functions, 163
expected value, **65**

factored form, **7**
fast convolution, **172**
flip operator, **148**
Fourier Dual, 173
Fourier theorems, **145**
frequency bin, **136**
frequency response, **181**
frequency-domain aliasing, **160, 161**
fundamental theorem of algebra, **11**

geometric sequence, **127**
geometric series, **127**

Hermitian spectra, 167
Hermitian symmetry, **167**
Hermitian transpose, **212**
hex, 54
hexadecimal, **54**

ideal bandlimited interpolation, **177, 178**
ideal bandlimited interpolation in the time domain, **178**
ideal lowpass filtering operation in the frequency domain, **178**
identity matrix, **214**
IDFT, **145**
imaginary part, **12**
impulse response, **180**
impulse signal, 180
indicator function, **171**
Intensity, **44**
intensity level, **48**
interpolation operator, **177**
inverse DFT, 1, **145**
inverse DFT matrix, **139**
irrational number, 30

JND, 45
just-noticeable difference, **45**

lag, **154**
lagged product, **154**
length M even rectangular windowing operation, **178**
linear combination, **88**
linear phase FFT windows, 171

linear phase signal, **171**
linear phase term, **170**
linear transformation, **214**
logarithm, **41**
loudness, **47**

magnitude of a complex number, **14**
main lobe, **134**
mantissa, **42**
matched filter, **153**
matrices, **211**
matrix, **211**
matrix columns, 211
matrix multiplication, **212**
matrix rows, 211
matrix transpose, 211
mean, 65
mean of a random variable, **65**
mean of a signal, **65**
mean square, **66**
mean value, **65**
modulo, **147**
modulus of a complex number, **14**
moments, **66**
monic, **7**
Mth roots of unity, **29**
mu-law companding, **64**
multiplication in the time domain, **173**
multiplication of large numbers, 42
multiplying two numbers convolves their digits, **154**

natural logarithm, **42**
negating spectral phase flips the signal around backwards in time, **167**
non-commutativity of matrix multiplication, 213
nonlinear system of equations, **8**
norm of a complex number, **14**
normalized *inverse* DFT matrix, 139
normalized DFT, **174**
normalized DFT matrix, **139**
normalized DFT sinusoid, **174**
normalized DFT sinusoids, **132**, **136**
normalized frequency, **146**

octal, **54**
odd functions, 163
orthogonal, **139**
orthogonality of sinusoids, 128
orthonormal, **139**

parabola, **9**
PCM, **53**
periodic, **146**
periodic extension, **134**, **146**
periodogram method, **186**
periodogram method for power spectrum estimation, **186**
phase response, **182**
phasor, **87**
phasor angle, **88**
phasor magnitude, **88**
phon amplitude scale, **48**
polar coordinates, **14**
polar form, **29**
polynomial approximation, **39**
power, **44**
power spectral density, **186**
power spectrum, **186**
pressure, **44**
primitive Mth root of unity, **30**
primitive Nth root of unity, **129**

quadratic formula, **9**

rational, **29**
real part, **12**
rectangular window, **136**
rectilinear coordinates, **14**
rms level, **66**
roots, **7**
roots of unity, **29**, **128**
row-vector, **212**

sample coherence function, **188**
sample mean, **66**
sample variance, **66**
sampling rate, **2**
second central moment, **66**
sensation level, **48**
shift operator, **148**
sidelobes, **134**
signal dynamic range, 52
similarity theorem, **188**
sinc function, **134**, **179**
sinc function, aliased, **134**
skew-Hermitian, **167**
smoothing in the frequency domain, **173**
sone amplitude scale, **48**
Sound Pressure Level, **47**
spectral leakage, **134**
Spectrum, **2**
spectrum, **132**, **145**
SPL, 47
square matrix, 211
standard deviation, **66**
stretch by factor L, **155**
Stretch Operator, 155
symmetric functions, 163
system identification, **184**, **188**

Taylor series expansion, **36**
time-domain aliasing, **160, 161**
Toeplitz, **214**
transform pair, **146**
transpose of a matrix product, **213**

unbiased cross-correlation, **183**
unit pulse signal, 180
unitary, **139**

variance, **66**

window, **134**
windowing in the time domain, **173**

Zero padding, **155**
zero padding, **177**
zero padding in the frequency domain, **156**, **178**
zero padding in the time domain, **156**
zero phase signal, **169**
zeros, **7**